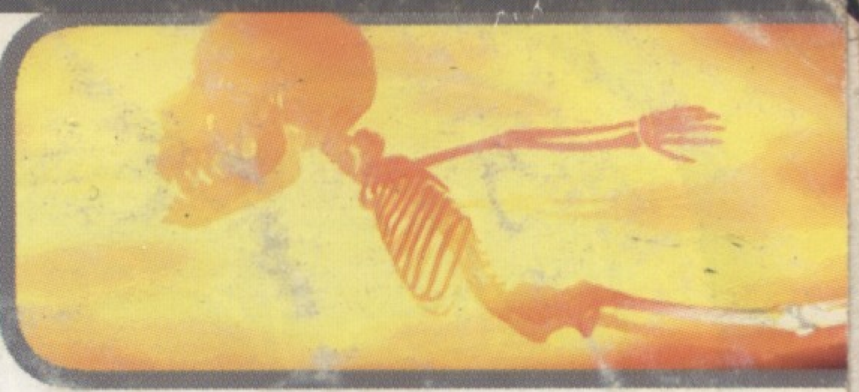
Premios para los lectores • Estrategia • C y Ensamblador • Aventura • DIV 2 • Arcade



CONCURSO JUEGOS DIV

DIV manía

995 ptas.
PORTUGAL 990 ESC (CONT) 5,98 €

www.prensatecnica.com
Año 1 • Número 4



• *Vida artificial*
Cómo imitar la vida real en la pantalla de tu ordenador

• *Diseño y dibujo*
La manera de diseñar objetos para tus juegos

• *DIV interno*
Por las entrañas de la programación

• *3D Red*
Creación de mapas 3D con el programa DIV 2

• *Webs DIV*
DIVnet, una revista electrónica sólo disponible en Internet

• *Programación*
Todo lo que debes saber para programar tus propios videojuegos

• *Share música*
Aprende a grabar la banda sonora de tus juegos

Cómo hacer que nuestro código trabaje a la velocidad del rayo

OPTIMIZAR PROGRAMAS

Prens@ Técnic
00004
8 413042 087533

DIVmanía

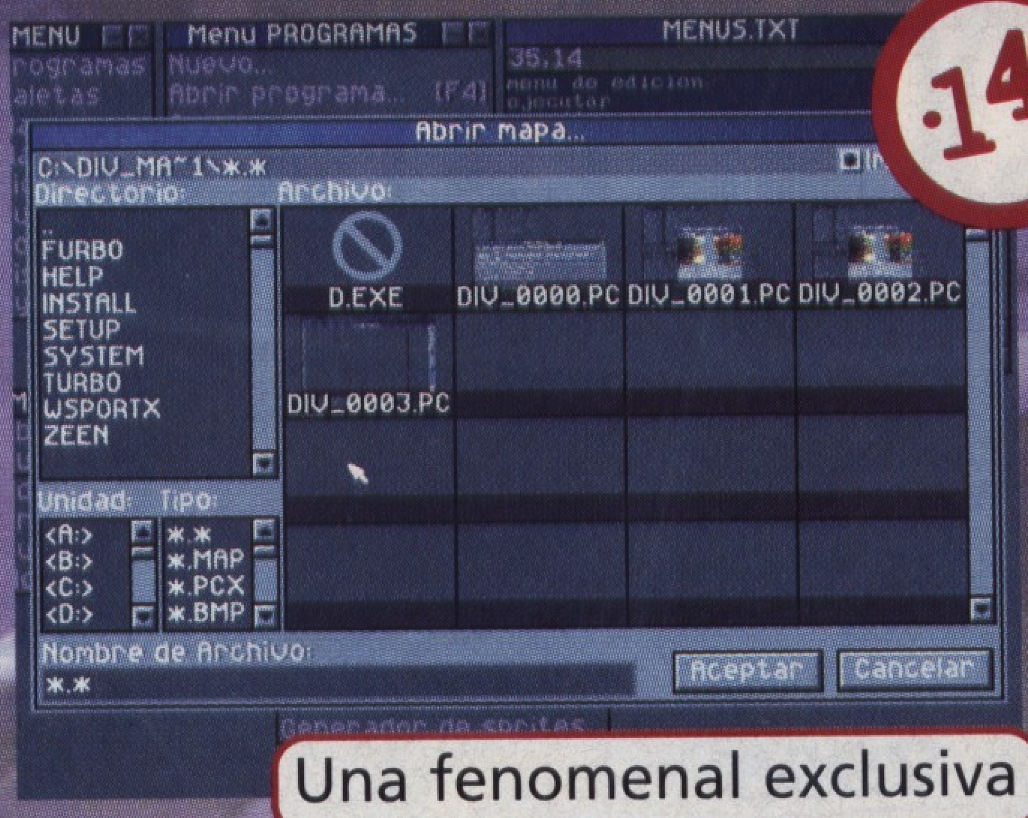


¿Amas la programación de juegos...? Léete estas líneas

Divmanía sigue al pie del cañón. Ya tenéis en vuestras manos el numero 4 de vuestra revista preferida, y seguro que mucho antes de lo que pensabais. Somos conscientes de los retrasos que ha tenido nuestra publicación y por ello pretendemos hacer un esfuerzo para que no se repitan. Esperamos, por tanto que a partir de ahora, Divmanía llegue a vuestras manos puntualmente.

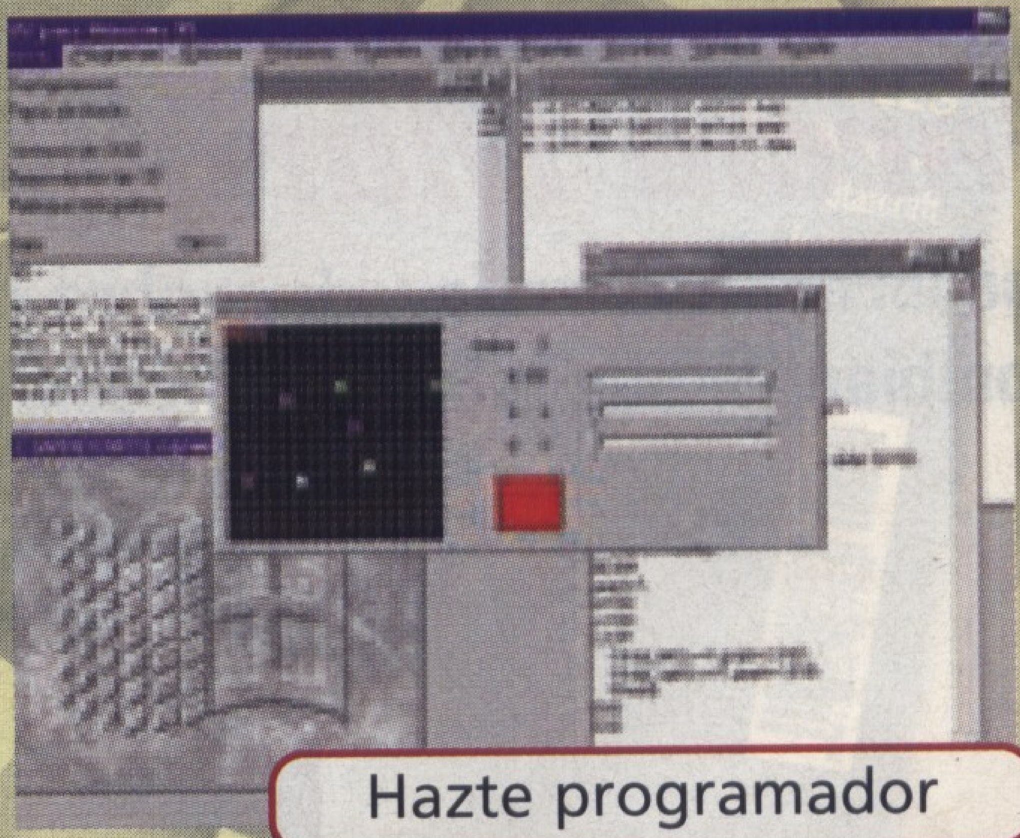
El número que tenéis en vuestras manos presenta algunos cambios que, sin alterar el contenido habitual de la publicación, sí son significativos. En primer lugar, nos hemos visto obligados a suprimir una sección por problemas ajenos a nosotros. Esperamos poder seguir contando con esta sección en siguientes números pero de momento hemos tenido que sustituirla. Sin embargo, hemos incluido un reportaje especial que seguro despertará gran interés entre todos vosotros.

Si en el número anterior os comentábamos que nos estábamos afianzando, en éste no podemos sino afirmar que nuestra revista se ha convertido en un punto de referencia fundamental sobre el que gira todo el mundo de DIV. Y la mejor prueba de ello es la aparición de revistas electrónicas exclusivamente dedicadas a este entorno que se han dirigido a nosotros para presentarse y pedirnos nuestro apoyo. Nosotros, por supuesto, no podemos negarnos y así lo hemos hecho. Todo para que el mundo DIVero se convierta en un entorno solidario y donde la colaboración y la ilusión de cientos de personas volcadas en la programación de juegos encuentren su lugar.


Una fenomenal exclusiva


Reportaje especial

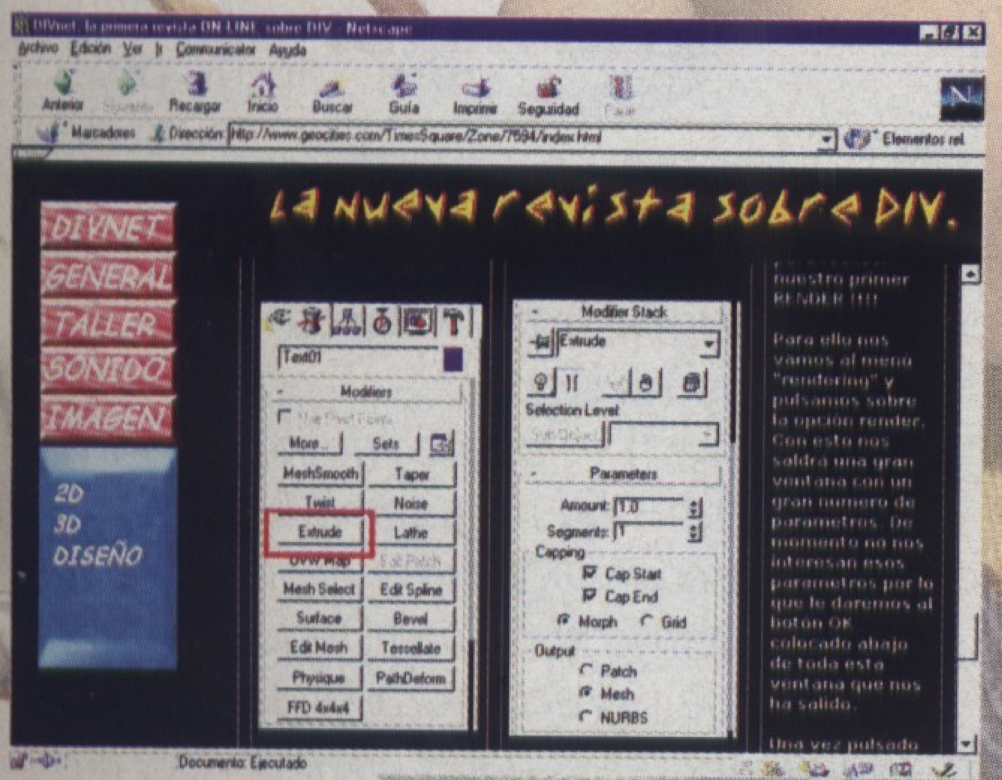
DIV Developer

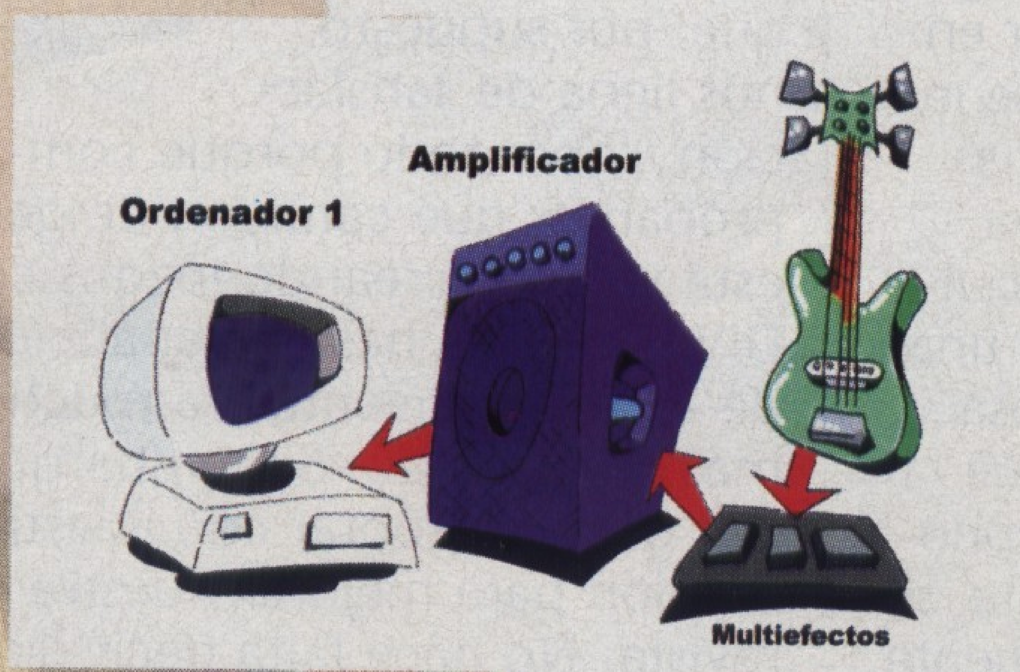

Hazte programador

Director: Mario Luis
mluis@prensatecnica.com
Coordinador Técnico: Oscar Condés
gover@prensatecnica.com
Colaboradores: Pablo Trinidad, José M. Sevillano, Sergio Cánovas, Miguel Barroso, David Martínez, Emilio Llamas, Javier Fernández, Fermín Vicente, Armando Vélez, Ramón de España, Carlos González y Jordi Martín.
Edición: Alfredo del Barrio y Martín Moncalvillo
Dirección de Arte: Francisco Calero
Maquetación: Manuel J. Montes, José A. Gil, Antonio Barbero, Ana Isabel Madero, Maribel Grande y Susana Burgaleta
Portada: Francisco A. Anguís
Publicidad: Marisa Fernández, marisa@prensatecnica.com
Sonia Glez.-Villamil, Noelia Menéndez y Emerio Arena
Supervisión CD-Rom: Alvaro García y Noé Soriano
Servicio Técnico CD-Rom: Eugenio García
Horario de atención: tardes 16:00 - 18:00 h
E-mail: stecnico@prensatecnica.com
Secretaria de Redacción: Elena Fernández
Departamento de Suscripciones: Sandra Fernández y Noemí Iscart
suscripciones@prensatecnica.com
Departamento de Administración: Juan López, Juan Ignacio Domínguez
Departamento Comercial: Marcelino Ormeño

REDACCIÓN, PUBLICIDAD Y ADMINISTRACIÓN
c/ Alfonso Gómez 42. Nave 1.1.2
Madrid 28037. España
Tfno: (91) 304. 06. 22
Fax: (91) 304. 17. 97
Si llama desde fuera de España, marcar (+34)

Sumario





Programas del lector

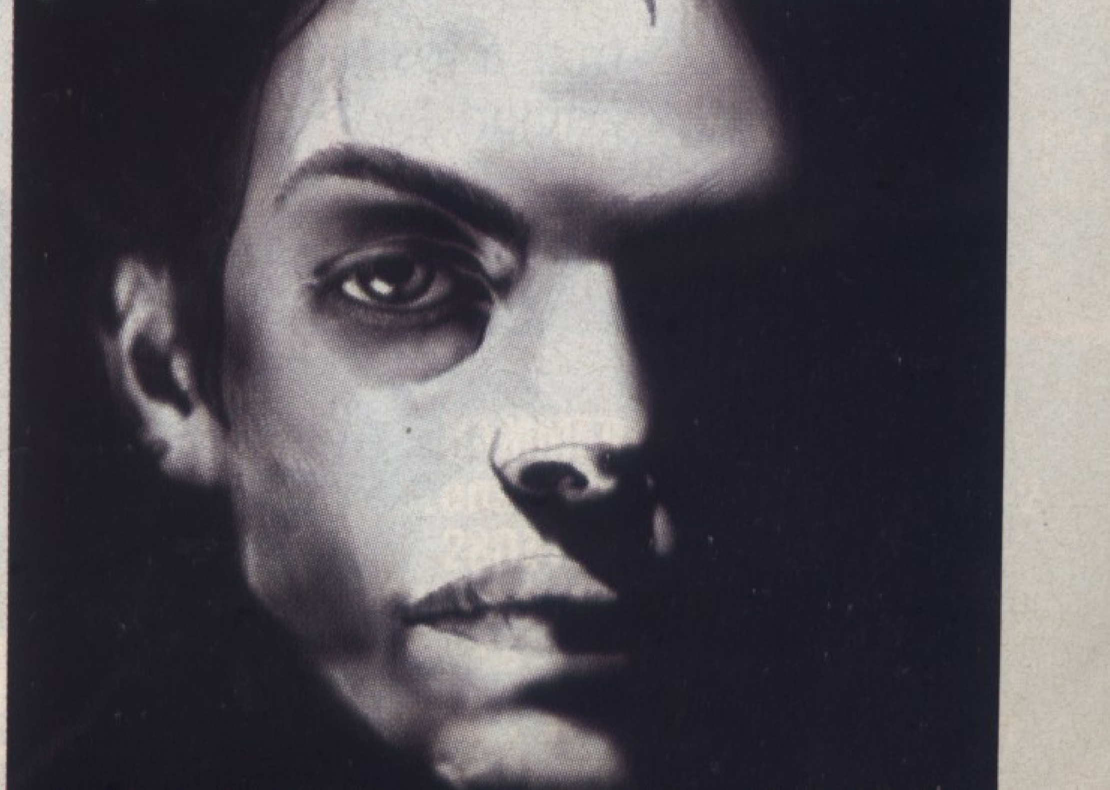


PREMIAMOS LOS MEJORES JUEGOS DE LOS LECTORES

DIV ha creado toda una escuela dentro del mundo de la programación. Todos los que se han acercado a la herramienta han sido capaces de programar. Por ello realizamos un concurso entre los lectores que hayan realizado juegos con DIV. Os ofrecemos un primer premio de 25.000 pesetas y dos accésit de 20.000 pesetas.

¿QUÉ ES DIV GAMES STUDIO?

DIV Games Studio es una herramienta de programación que facilita en gran manera nuestra inmersión en el software de entretenimiento. Es el primer entorno profesional que permite realizar videojuegos con fines comerciales sin necesidad de un pago adicional. Con el carné de desarrollador incluido se permite el desarrollo de cualquier tipo de juegos y su libre venta y distribución.



E-mail: ntactual@prensatecnica.com
http://www.prensatecnica.com
Horario de atención al público: de 09:00 a 19:00 h ininterrumpidamente
EDITA: PRENSA TÉCNICA
Director General: Mario Luis
Director Editorial: Eduardo Toribio
Director Técnico: Fernando Escudero
Director de Producción: C.P. Cerezo
Directora Publicidad: Marisa Fernández
Director Comercial: Esteban Martínez
Fotomecánica: Grafoprint
Impresión: Pantone S.A.
Duplicación del CD-Rom: M.P. O., Servicios Ibéricos, Grupo Cóndor
Distribución: SGEL. Avda Valdelaparra, 29 Alcobendas. Madrid

Depósito legal: M-42077-1998
AÑO 1 • NÚMERO 4
Copyright 30-01-00 – PRINTED IN SPAIN

El lector lúdico

Sólo para aficionados a los juegos

Como bien sabéis, este pequeño espacio está dedicado a vosotros, los lectores, que al fin y al cabo sois la base de esta revista. Por ello, tratamos de reflejar aquí todas las opiniones que surgen alrededor de nuestra publicación.

Un número más estamos en la calle a pesar de las dudas de algunos. Parece ser que incluso se ha llegado a calificar a Divmanía como "Revista fantasma". Tal vez sea cierto que, hasta ahora, la periodicidad no ha sido la que deseábamos, pero estamos convencidos de que este apelativo no tiene ninguna justificación. Hay que tener en cuenta que partimos de una serie de circunstancias extraordinarias que han hecho difícil mantener una estricta regularidad; especialmente la dificultad de realizar una revista absolutamente novedosa, basada en una herramienta que hasta ahora era bastante desconocida, aunque, afortunadamente, se está dando a conocer rápidamente. En este sentido somos como pioneros en un terreno inexplorado y lleno de vicisitudes, las cuales, gracias a nuestro esfuerzo e ilusión, estamos tratando de sobrellevar. Todo con el fin de llevar a nuestros lectores una revista para conocer el mundo DIVero. Todo para que los cientos de personas que han encontrado en DIV la posibilidad de crear sus propios juegos, algo que hasta la aparición de este entorno estaba reservada a los que tenían conocimientos de programación, tengan unas páginas donde aprender, informarse e incluso intercambiar ideas.

Quisiéramos, desde aquí, animar a los lectores a que nos envíen (por cualquier medio) sus opiniones, dudas, comentarios, etc. Lo cierto es que la mayoría de la correspondencia que recibimos es para participar en el concurso de programación, algo que, por supuesto, nos llena de satisfacción, sobre todo porque comprobamos qué cantidad de gente está volcada creando juegos con DIV, algo que hasta la aparición de esta herramienta era mucho más complicado. Sin embargo, nos gustaría contar con vuestra opinión para mejorar nuestra revista. No hace falta decir que ésta es una revista abierta, tanto a la opinión como a la participación. Cualquiera que tenga algo que decir puede ponerse en contacto con nosotros y aquí encontrará su "huequecito".



En este caso vamos a hacernos eco de algunas cuestiones que han surgido en torno a nuestros colaboradores. Cuestiones de bastante importancia sobre todo si tenemos en cuenta que han surgido de las dudas que nuestros lectores han hecho llegar a los responsables de cada uno de nuestras secciones:

He recibido un correo de una persona que no encontraba los fuentes del curso de aventuras gráficas de DIV Manía 2. Curiosamente, yo ni me había dado cuenta de ese detalle. Cuando leo la revista me da por leer mi artículo y ver cómo ha quedado, pero no se me había ocurrido mirar las fuentes en el CD. ¿Podéis introducirlas en el CD de otro número? Miguel Angel Barroso.

No hay problema, las fuentes a las que te refieres, el artículo de programación de aventuras del número 2 de Divmanía, las encontraréis en el CD del presente número.

En la sección Web Div de Divmanía han aparecido noticias y curriculums de mi página web pero no apareció el nombre de la página y sin embargo sí de otras que llevan años que sin actualizarse. Quería saber si podéis hacer algo porque en el último número aparecieron curriculums pero sólo partes donde decia "si alguien necesita ayuda me ofrezco a ayudarle en sus juegos", de modo que todos los que aparecimos hemos recibido montones de e-mails (¡yo no puedo hacer 300 juegos a la vez!). En fin, que lo único que quiero es que aparezca el nombre de mi página, http://pagina.de/ferminho, para que la gente pueda acudir, ver los curriculums al completo y enterarse bien de la oferta. Gracias. Ferminho.

Nada más, esperamos que os guste el presente número y que el siguiente llegue a los kioscos a su debido tiempo. Por nuestra parte os podemos asegurar que haremos lo imposible para que así sea.




a uno de

o de una
aba los
nturas grá-

había dado
uando leo
mi artículo
pero no se
r las fuen-
éis introdu-
de otro
iguel
arroso.
ay pro-
ema, las
fuentes a
las que te
refieres,
el artícu-
o de pro-
mación de
de
eis en el

le
oticias y
web pero
a página y
e llevan
Quería
orque en
on curricu-
e decia "si
ofrezco a
modo que
emos reci-
yo no
a vez!). En
es que
ágina,
para que
os curricu-
e bien de

que os
y que el
os a su
a parte
haremos
ea.



Libertad sin cables



Este es el lema de la conocida empresa fabricante de periféricos diseñados para establecer un vínculo intuitivo y natural entre el usuario y el ordenador personal. Logitech presentó en Madrid su nueva gama de productos. Un total de 22 nuevos productos que vienen a completar una amplia gama de periféricos entre los que destacan los conjuntos de teclado y ratón inalámbricos, además de los gamepad o volantes destinados especialmente para los jugones.

Los nuevos productos Logitech responden a tres claves: la ya mencionada libertad sin cables, una fuerte apuesta por Internet y un diseño único y exclusivo de sus productos. La idea de Logitech es hacer más cómoda la relación de las personas con su ordenador y en ello su apuesta por la tecnología sin cables, vía radio-frecuencia, es el punto que más les define. Así, su gama de ratones inalámbricos se ha visto ampliada notablemente.

En lo que se refiere a los videojuegos, Logitech apuesta claramente por la creación de periféricos especialmente pensados para nosotros. La mayoría de ellos utilizando la ya conocida tecnología *Force Feedback* para lograr transmitirnos las sensaciones del juego. En esta línea, se ha desarrollado un ratón expresamente diseñado para jugar, *Wingman Gaming Mouse* (con el que, por cierto, regalan el juego *Shogo*), que permite mayor rapidez y precisión para nuestras partiditas, un gamepad, *Wingman Gamepad Extreme*, que responde a los movimientos de nuestra muñeca y hace innecesario el uso de los botones del cursor, un joystick, *Wingman Extreme Digital 3D*, especialmente diseñado para los amantes de la simulación, y un volante, *Wingman Formula Force*, que pretende hacernos sentir como si estuviéramos a los mandos de todo un Fórmula 1.

Las compañías premiadas en el ETCS





A la feria más importante a nivel europeo del sector de los videojuegos, conocida como ETCS, van muchas compañías pero sólo unas cuantas elegidas se hacen con los premios que otorga el jurado de este certamen.

Las compañías del mundo entero presentan sus últimos trabajos durante el desarrollo de esta exposición. A los mejores títulos se les da el reconocimiento que merecen y se les inscribe en el cuadro de ganadores de ese año.

En la edición del ETCS de 1999 han faltado algunas empresas importantes del sector de la programación del software de entretenimiento. Pero las presentes eran muchas y todas poniendo de largo sus últimas novedades. El sueño de cualquier aficionado a los videojuegos.



Os vamos a dar a continuación el cuadro de honor de este año.

La empresa Konami ganó el premio inaugural del ECTS por la segunda parte del juego *ISS Pro Evolution*. Además ganaron el premio al mejor juego para la plataforma PlayStation.

El mejor título para PC de la feria fue para el juego *Grand Prix 3*, de la empresa Hasbro Interactive.

En el apartado de juegos para la consola Nintendo el premio fue conseguido por *Donkey Kong 64*.

Ready 2 Rumble, se llevó la corona como mejor juego para la consola Dreamcast de Sega.

Y finalmente en la sección de juegos para multijugador se llevo el gato al agua *Rainbow Six: Rogue Spear.* Nuestra enhorabuena a los premiados desde estas páginas.



Sony usará tecnología RTIME en la PlayStation 2



Sony ha confirmado recientemente que su nueva tecnología RTIME permitirá un rápido desarrollo de aplicaciones multijugador para su nueva consola PlayStation 2. Según la empresa Sony, las mejoras más palpables que traerá la nueva tecnología RTIME incluyen un perfeccionamiento considerable entre las conexiones por módem ya que estas serán mucho más rápidas que en la actualidad.

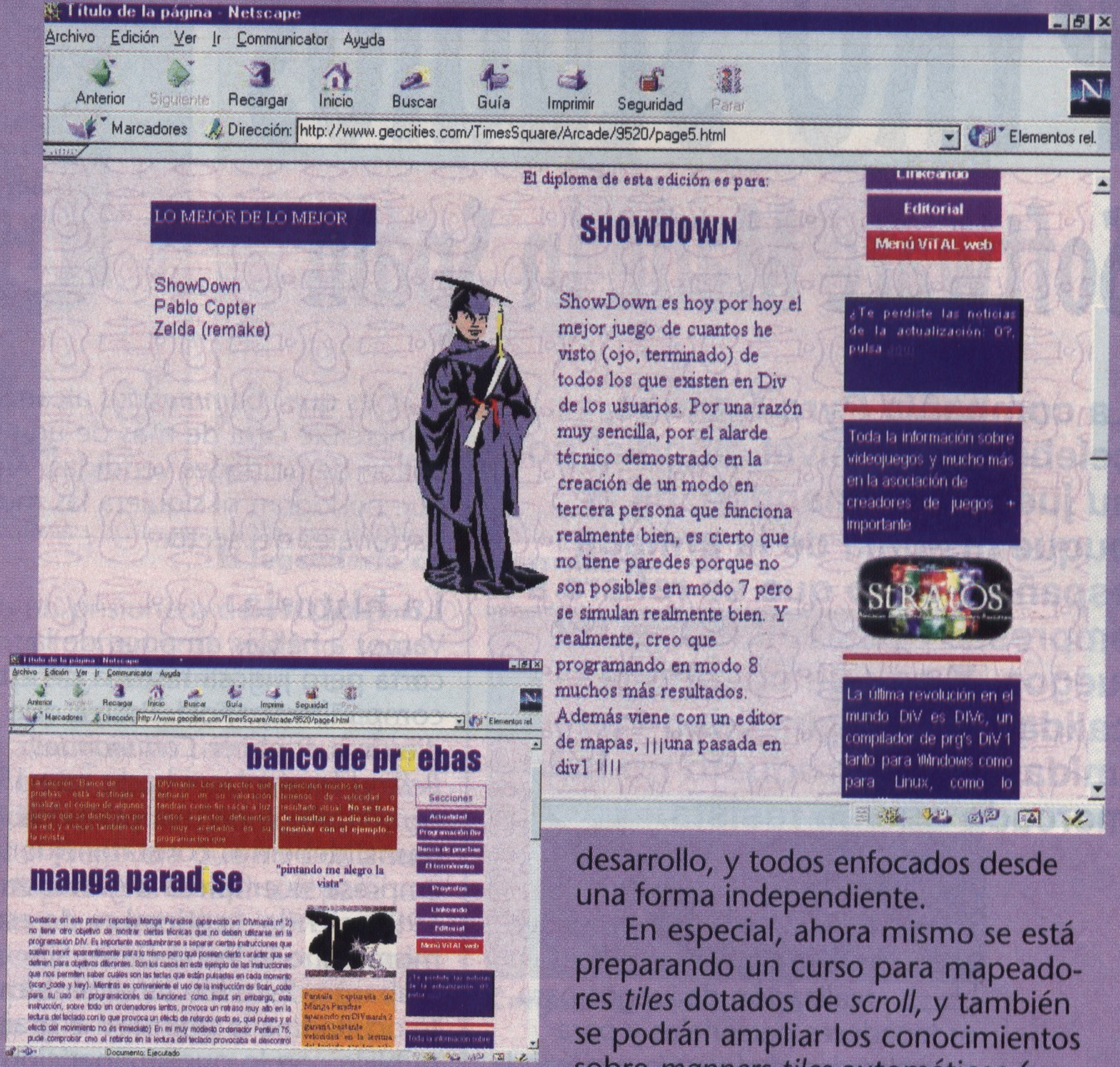
Además, con este software los desarrolladores se pueden aprovechar de las últimas tecnologías y pueden mejorar la experiencia del usuario ofreciendo voz, música, vídeo, y gráficos en tiempo real.

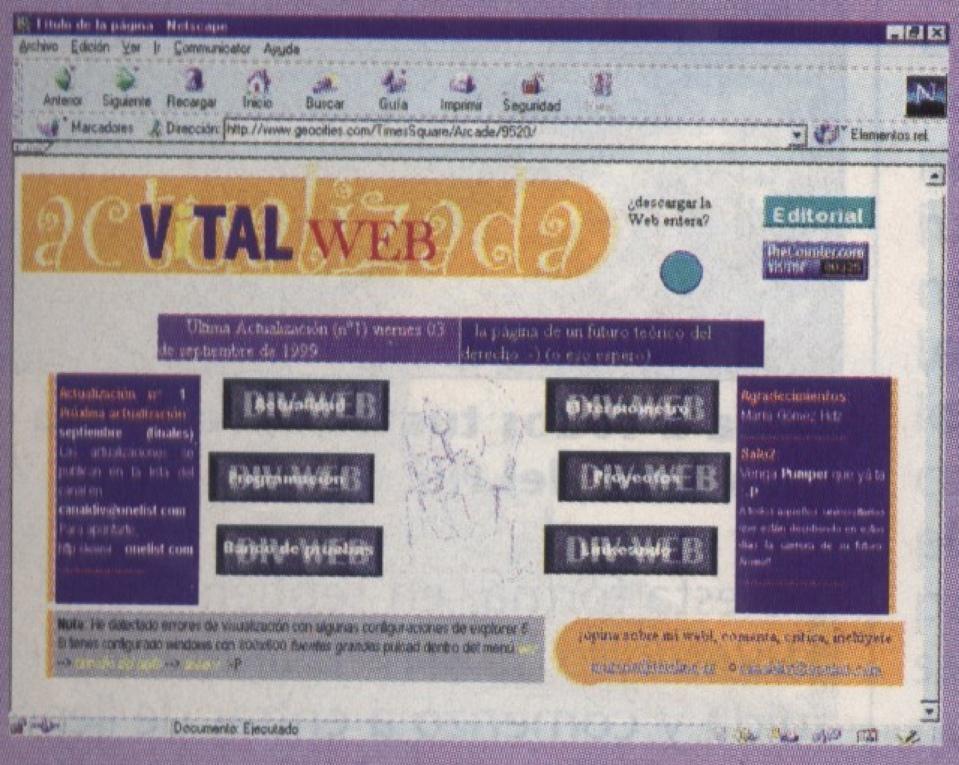


VITAL web

Aunque en el reportaje que aparece en las páginas interiores de esta revista aseguramos que DIVnet es la única revista electrónica dedicada al entorno *DIV*, parece que Internet va más deprisa que nosotros. Hemos recibido información de una segunda publicación en Internet dedicada al mismo tema: el mundo *DIV*. Aquí tenéis cumplida información de ella, aunque sea brevemente. Esperamos ocuparnos de esta revista más extensamente en el siguiente número de DIVManía.

Parece que la nueva publicación electrónica no se morderá la lengua a la hora de criticar nuestra *Div Manía*, y todo lo relacionado con el programa *DIV*. Toda crítica o sugerencia de nuestros lectores será siempre bien recibida en esta redacción. Y ahora pasamos a desgranar lo que es y será ViTAL web.





ViTAL web será una página dedicada a todo lo relacionado con el entorno *DIV*, aunque estarán abiertos a la aparición de nuevas herramientas de desarrollo de videojuegos. También se intentará cubrir un amplio espectro de temas, entre ellos los dedicados al desarrollo, y todos enfocados desde una forma independiente.

En especial, ahora mismo se está preparando un curso para mapeadores *tiles* dotados de *scroll*, y también se podrán ampliar los conocimientos sobre *mappers tiles* automáticos (un programa dedicado a aquellos que no sepan utilizar los mapeados *tiles* en *scroll* o que simplemente quieran agilizar sus tareas).

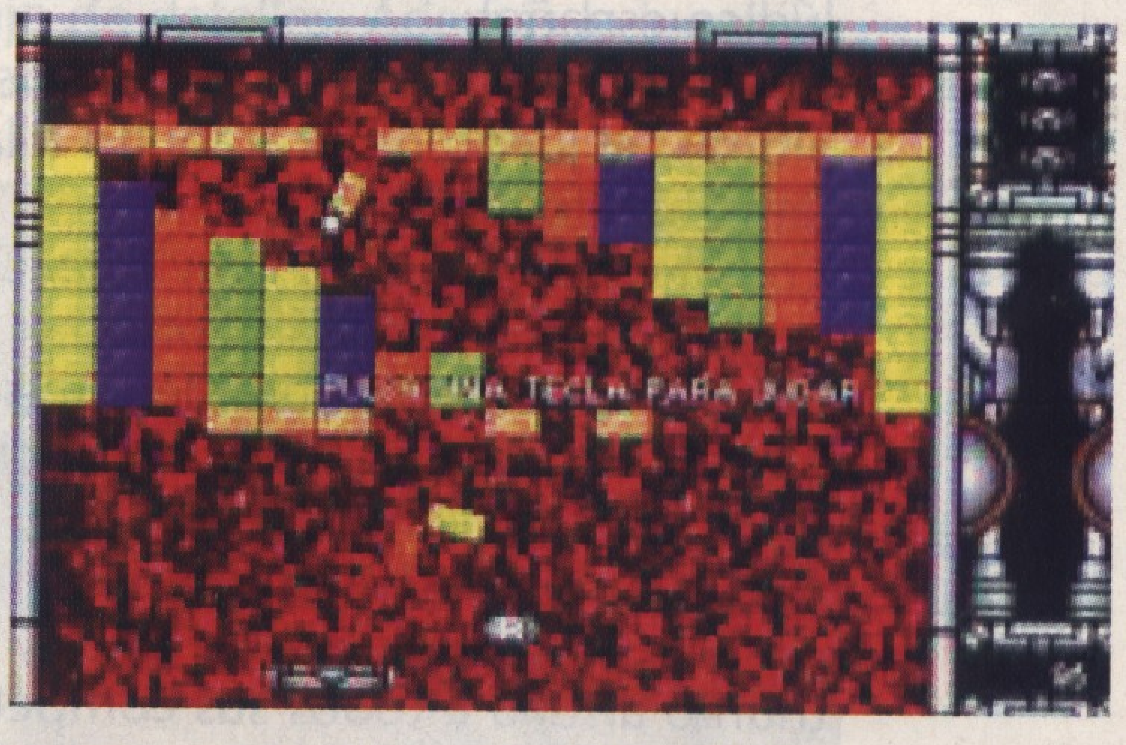
Esperamos que visitéis pronto esta nueva dirección: *http://pagina.de/vital* o *http://pagina.de/vital-web*

Aparece un compilador de DIV



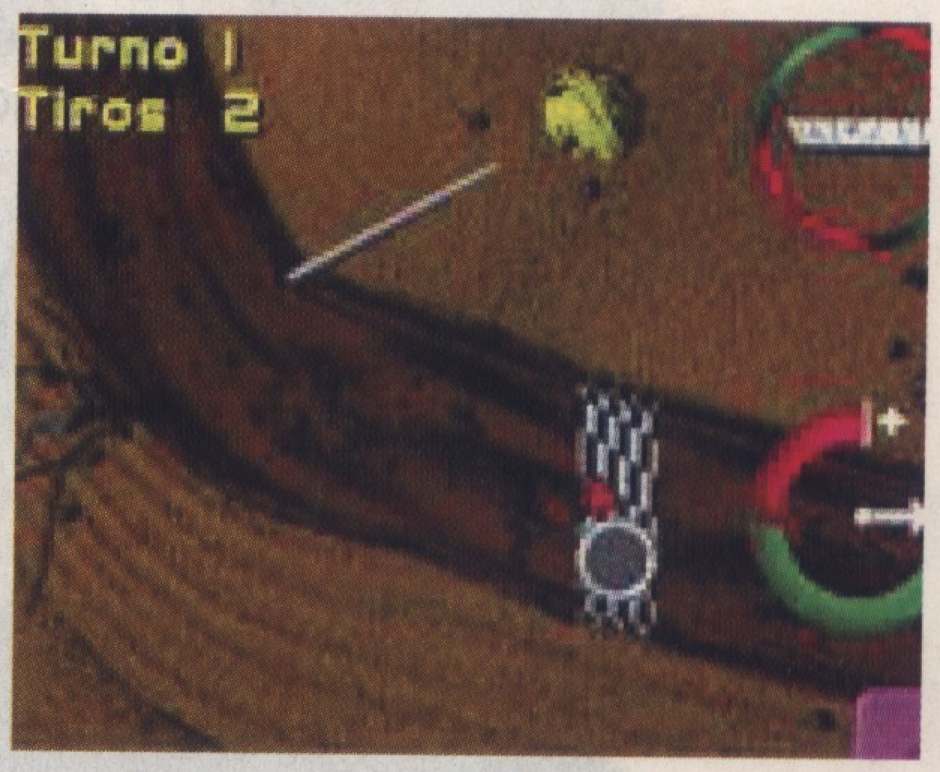
El nombre de este compilador/interprete del programa *DIV* es DIVC y podéis encontrarlo en la página de su creador sita en la siguiente dirección de Internet: *http://www.arrakis.es/~jlceb/divc*

Lo que permite hacer DIVC es poder ejecutar los juegos realizados con el programa *DIV Games Studio* en plataformas distintas a *MS-DOS*. Para lograrlo interpreta el contenido de los ficheros de código fuente de *DIV* (*.prg) y los compila a un formato interno (*bytecode*) que luego pasa a ejecutar con la ayuda de una librería que duplica la funcionalidad existente en el programa *DIV Games Studio* en su versión 1. DIVC es un totalmente *freeware*, o sea que podéis bajarlo y usarlo libremente, incluso su creador permite que, si queréis, modifiquéis el código para hacer versiones propias. Eso sí, si lográis hacer una versión de este programa deberéis distribuirla de la misma manera.





Este software es todavía un proyecto en desarrollo, aunque se puede usar la versión beta del mismo. En la página arriba indicada tendréis toda la información necesaria para seguir los pasos previos al nacimiento de este compilador de *DIV*.



PYRO Studios

Software Made in Spain



La compañía Pyro Studios, celebérrima a nivel mundial por su juego Commandos, es el buque insignia de la armada española en lo que se refiere a empresas desarrolladoras de juegos. Y es que cuando la calidad y la originalidad van unidas en un producto no hay mercado que se resista.

La compañía Pyro Studios fue creada con un objetivo muy definido: abordar la realización de juegos para ordenadores personales y para consolas con títulos que tuvieran la calidad suficiente como para competir en todos los mercados internacionales. No sólo han logrado este ambicioso objetivo en un tiempo record, sino que se han convertido en uno de los desarrolladores de más éxito a nivel internacional, son conocidos en el mundo entero, y se han convertido en los abanderados del software lúdico español.

Como todos sabréis a estas alturas, el juego que catapultó a la fama a Pyro llevó por nombre *Commandos: Behind Enemy Line.* De lo que quizá no tengáis conocimiento es que esta obra de arte se situó en el número 1 de las principales listas de ventas del mundo durante más de 10 semanas.

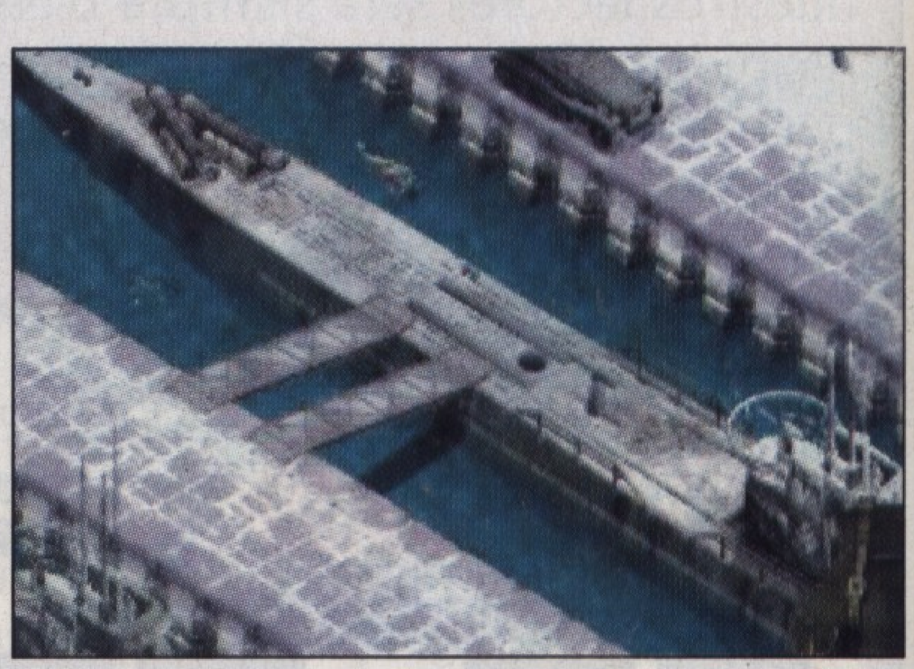
La fulgurante riada de éxito y triunfo arrolló a todos sus competidores e inundó de títulos vendidos a países como Alemania, España, Estados Unidos, Reino Unido, Francia, Italia, Holanda, Bélgica, Austria, Polonia, y un largo etcétera. Y es que *Commandos* alcanzó la increíble cifra de más de un millón de unidades vendidas. Algo que no logran ni siquiera las multinacionales del sector.

La historia

Vamos a hablar un poco de la corta pero jugosa historia de esta compañía española. Pese a que no dieron a conocer *Commandos; Behind Enemy Lines* hasta que 1998 comenzaba a dar sus primeros pasos, debemos constatar que la empresa se empieza a gestar en 1996, siendo septiembre de ese mismo año la fecha clave. Sin embargo, Pyro fue inteligentemente ocultada a la prensa y al gran público con lo que su aparición pilló por sorpresa a todo el mundo, no sólo por la calidad de su tarjeta de presentación, *Commandos*, sino por todo lo que supuso este videojuego para la historia del software nacional.

Al contrario de lo que pudiera parecer, Pyro no estaba formada a partir del tópico aquel de "jóvenes desconocidos, con mucho talento e ilusión, se reúnen un buen día para hacer un juegazo". No, Pyro fue una compañía que, desde el principio, se creó con el objetivo de producir juegos de alta condición, es decir, con calidad internacional y capaces de competir por los Top 10 en los países más importantes del mundo. Proein fue la compañía que se encargo de hacer factible tal proyecto. Aquí hay que citar la amplia capacidad de trabajo y de decisión de Ignacio y Javier Pérez Dolset, máximos responsables de que Pyro pudiese ser realidad como empresa, para luego proporcionar esa infraestructura tan necesaria para las grandes compañías.

Sin embargo, una empresa de las características de Pyro necesitaba, sobretodo, de experiencia. Aquí es donde entrará en juego Gonzalo Suárez, antiguo miembro de Ópera Soft, que trajo bajo el brazo un buen montón de gente capaz de trabajar a tan alto nivel.



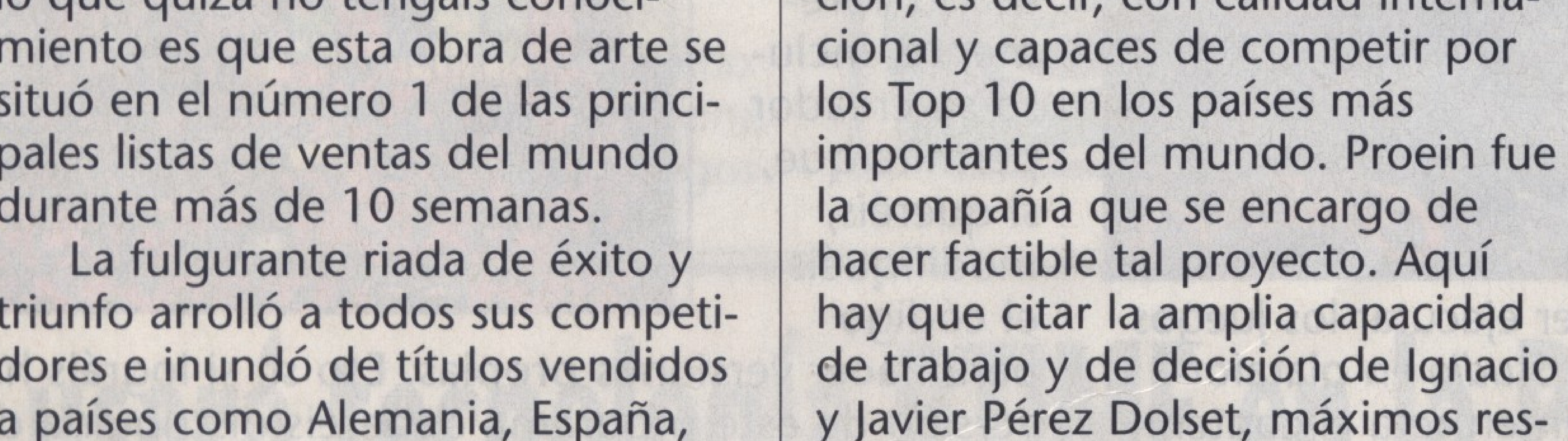
Lleva a todos tus hombres hasta el otro lado del río.

De esta forma, en septiembre de 1996, Pyro Studios quedó constituida y comenzó a entrar de lleno en la producción de *Commandos.*

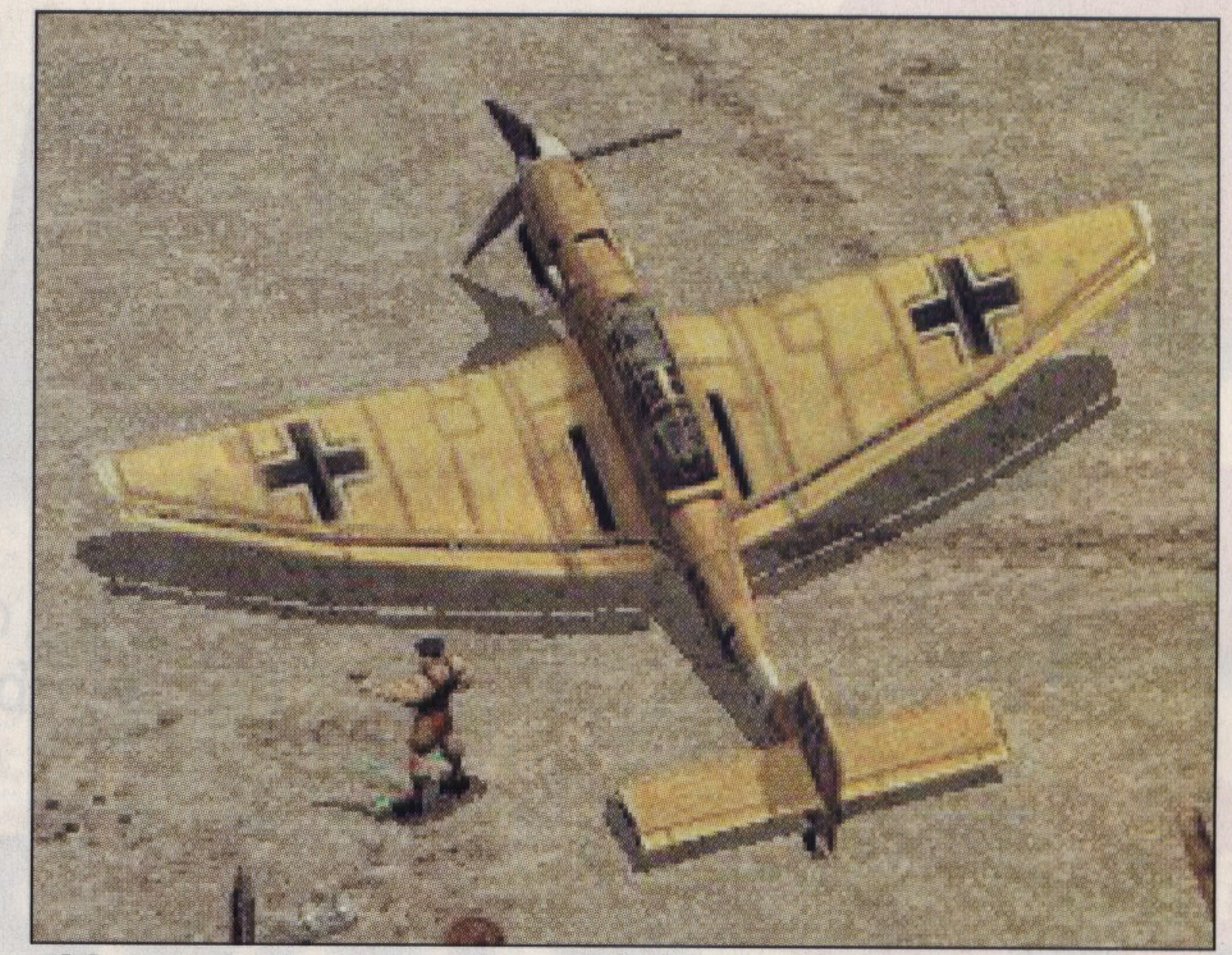
Pero no fue hasta febrero del 98, aproximadamente, cuando este proyecto comenzó a filtrarse a la prensa. Pronto empezó a armarse un gran revuelo en torno al mismo, dada la calidad aplastante de las capturas mostradas. Éstas eran la prueba más palpable de que en la cocina de Pyro Studios se estaba cociendo algo realmente espectacular. Y dado el enorme misterio que giraba en torno a su jugabilidad y concepto, que iba a suponer una renovación para el mercado de juegos de estrategia, *Commandos* terminó por convertirse en una "bomba de relojería".

En torno a Pyro se fueron congregando grandes nombres del mundo de la programación de nuestro país. Personajes como Jon Beltrán Heredia, Javier Fáfula, César Astudillo y Javier Arévalo, o grafistas tan prestigiosos como Jorge Blanco o Asier Hernáez, se dieron cita para configurar el equipo de desarrollo más importante de España.

Una cosa está clara, con *Commandos* se abre definitivamente




Las defensas antiaéreas están dispuestas.


El legendario caza Messrmitch.

una etapa nueva de hacer juegos en España; más profesional y, sobre todo, exportable. Hoy quien se dedique a producir este tipo de productos con vistas al mercado nacional exclusivamente está irremediablemente condenado al fracaso más absoluto o, como mucho, a la pura anécdota regional.



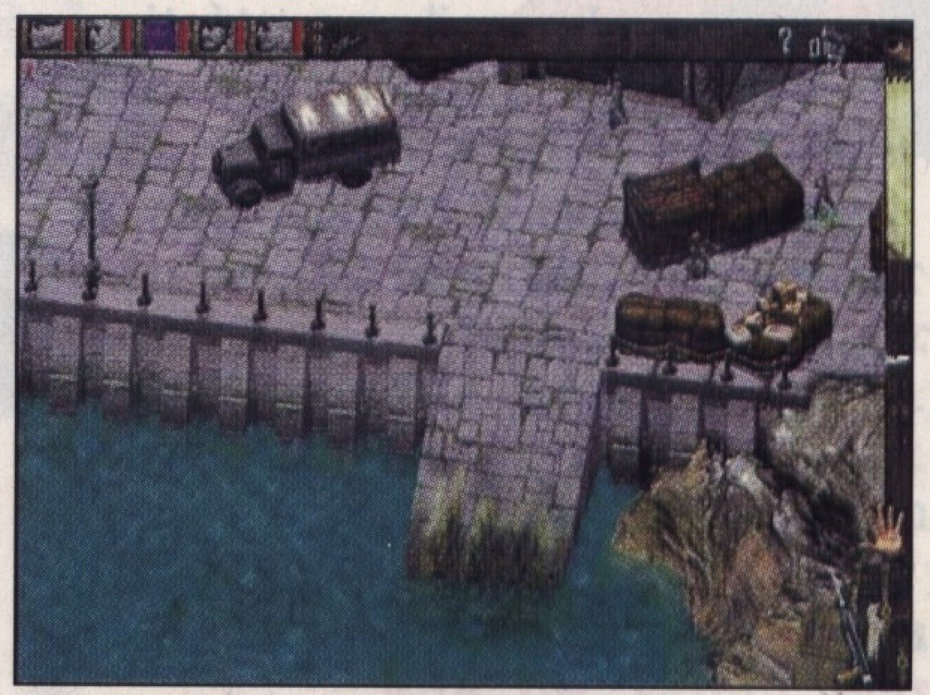
Pero lo más importante de todo, independientemente de lo bueno que sea *Commandos*, es la repercusión que Pyro va a tener en el futuro de la producción de videojuegos en nuestro país. Con su estupendo trabajo han demostrado que merece la pena confiar en productos 100% españoles, y gracias a ellos, pronto habrá más empresarios que decidan apoyar, de forma importante, nuevos proyectos de grupos nacionales de programación.

Commandos

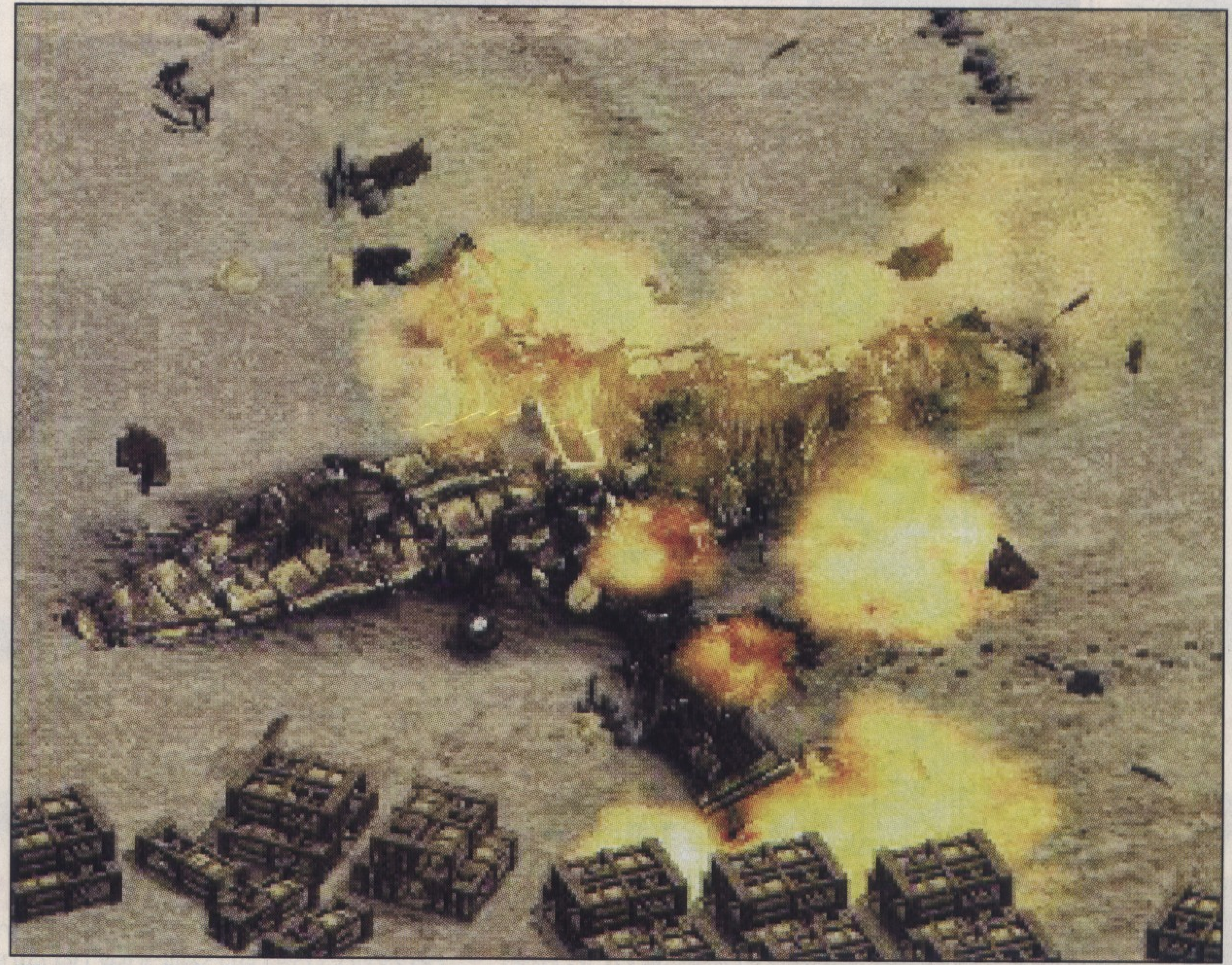
Vamos a hablar un poco de que características tiene el que es, hasta ahora, el juego español más famoso fuera de nuestras fronteras.

Commandos, Behind Enemy Lines introdujo un nuevo género dentro de los juegos para PC: la estrategia táctica. El juego consistía en hacerse cargo de un aguerrido comando aliado que debía realizar determinadas misiones detrás de las líneas alemanas durante la Segunda Guerra Mundial. Las misiones consistían en volar puentes, matar a altos mandos del ejército nazi, liberar a un prisionero de un campamento fuertemente vigilado, volar una presa, descarrilar un tren, acabar con las instalaciones militares que albergan las temidas bombas V-2, en fin pequeñas chapuzas de la vida cotidiana. Pero lo más original del juego era que no podíamos ir y disparar sin ton ni son para conseguir nuestros objetivos, eso no servía de nada en unos escenarios plagados de enemigos. Lo que había que hacer era planificar la misión paso a paso y aprovechar las distintas habilidades que tenían los hombres encargados de llevar a cabo la dura tarea. Un fallo en el plan de acción y las alarmas se disparaban, todo el ejército alemán salía a perseguirte y era inevitable que pasaras a engrosar la numerosa lista de héroes caídos en combate. Muertos con honor pero muertos al fin y al cabo.

Para sobrevivir a una misión había que hacer verdaderas proezas colocando a tus hombres en el sitio justo, arrastrándolos por el suelo, eliminando a los soldados de guardia en el más estricto silencio, poniendo trampas, etc. Lo más parecido que se ha visto a una guerra de verdad.


Cargad las provisiones.

Y no sólo era original el juego, es que además era de una gran calidad gráfica. Con unas pantallas espectaculares realizadas con todo lujo de detalles. Además la jugabilidad era altísima, cada escenario


El arsenal del enemigo vuela por los aires.


El puerto está controlado.


¡Acabad con los barracones del campo!

requería probar mil y una tácticas hasta dar con la adecuada para llevar a cabo todos los objetivos de las misiones. Las últimas pantallas tienen una dificultad que no conseguía pasarlas ni el mismo Rambo.

En resumen, una producción soberbia, que no sólo era el desarrollo más internacional hecho en España hasta la fecha, sino que pronto alcanzaría los Top 10 en los países más importantes del mundo, convirtiéndose más tarde en el tercer juego más vendido de todo el año 98.

Presente y futuro de la compañía

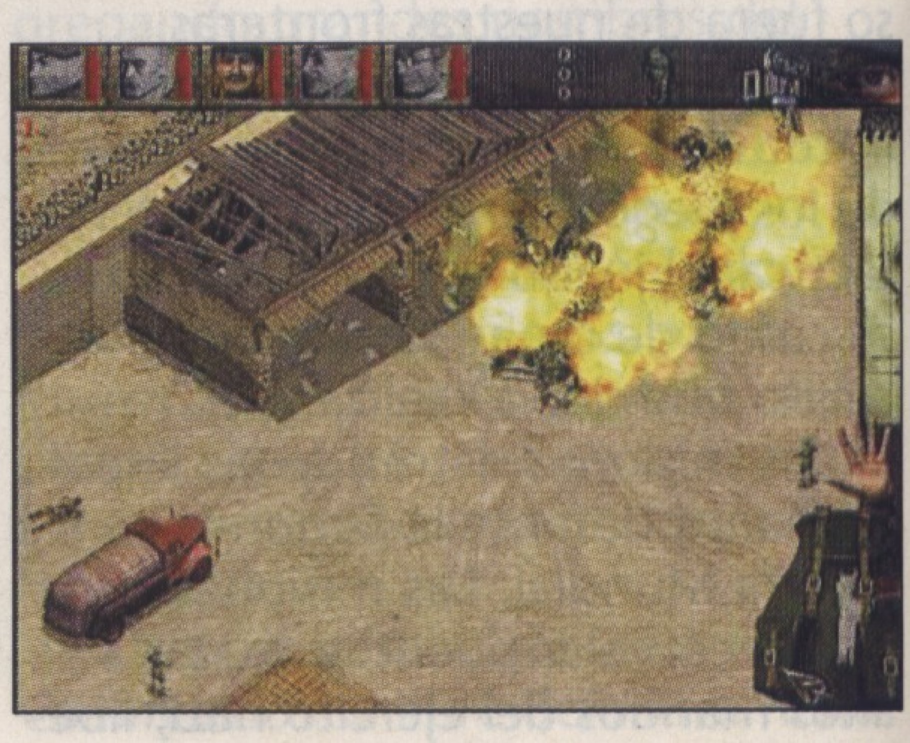
Ya con el marchamo del éxito a sus espaldas y con una aureola de notoriedad ganada en buena lid, Pyro editó a mediados del 99 el disco de expansión de su título estrella. Llevo por nombre *Commandos; Más Allá del Deber*.

En estos momentos Pyro Studios está desarrollando productos para todas las plataformas de videojuegos: PC, Sony PlayStation y Sega Dreamcast. La plantilla de la empresa supera actualmente las 70 personas. A este paso y con un par de buenos títulos más posiblemente estaremos hablando dentro de poco de la multinacional Pyro Studios.


Y este par de títulos pueden estar al llegar. Parece que ya se están desarrollando, aparte de las esperadas conversiones a consolas del juego *Commandos,* los juegos que serán el mascarón de proa de Pyro durante el año 2000. Las dos obras a que nos referimos permanecen en el más absoluto secreto, aunque fuentes de confianza nos han dicho que uno de los juegos será la segunda parte de *Commandos*, cosa que era de esperar. Hay que exprimir el filón que supone disponer de un nombre ya consagrado en el mercado. *Commandos* se convertirá en serie, cosa que agradecerán sus numerosos seguidores. Del otro juego no se sabe nada en absoluto. Sólo ha podido trascender que ahora mismo en la compañía hay dos grupos de desarrollo independientes, uno de ellos dirigido por Javier Fáfula y el otro por César Astudillo. Es de suponer que un grupo se estará encargando de la producción de *Commandos 2*, del otro grupo de desarrollo nada sabemos aunque probablemente su trabajo saldrá a la luz en el año 2000.

Bueno, nada más por ahora, sólo decir que en la página *web* de Pyro Studios hay un apartado en el que se piden colaboradores para entrar a formar parte de la plantilla de la empresa. Y es que no paran de crecer.






Hay que adueñarse de la casa y sellar todas las entradas.

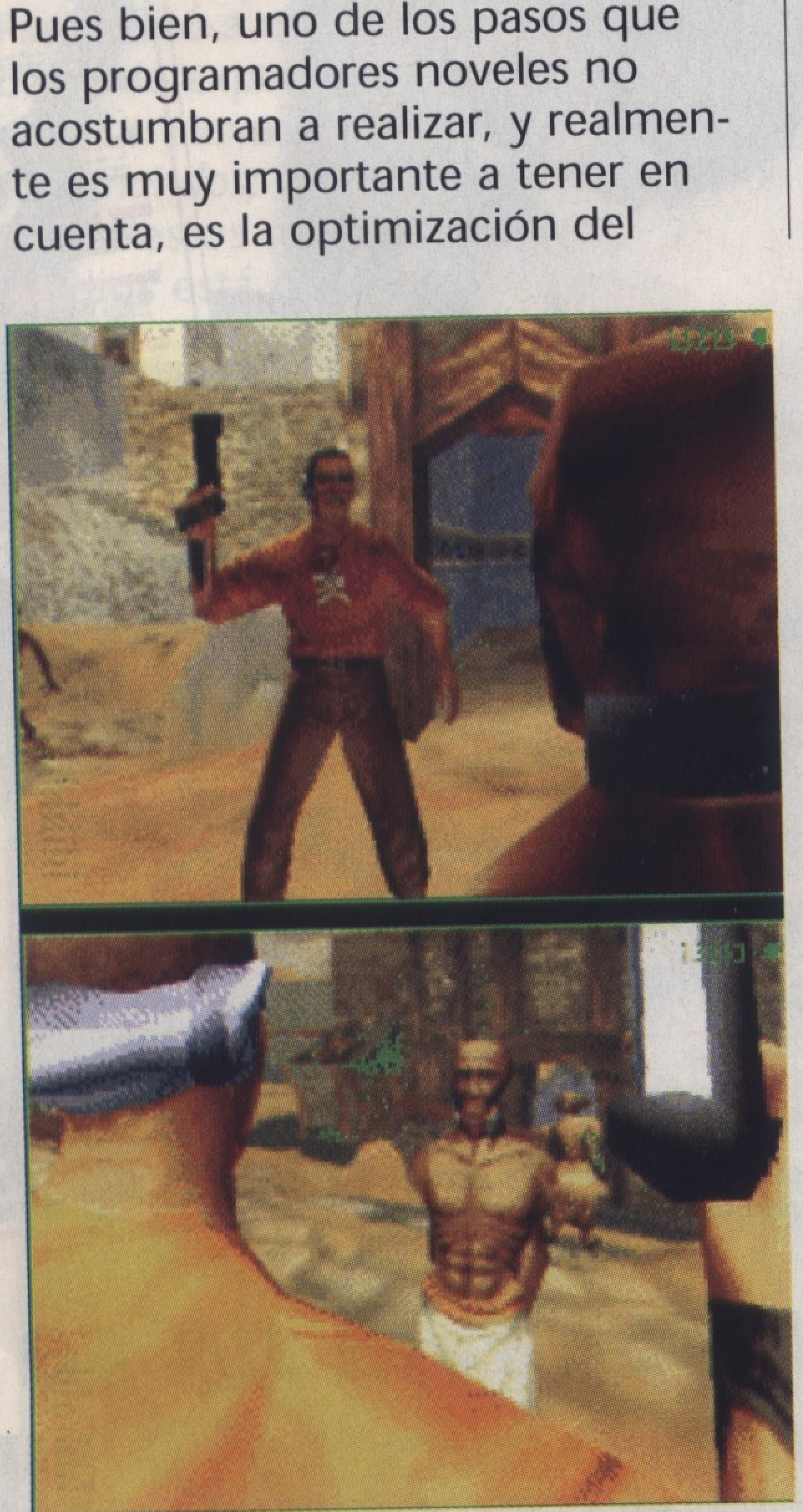

www.divgames.com
DIV 2
Juegos comerciales con DIV 2
Editor de sonidos.
Laberintos 3D.
Editor de mapas 3D.
Juegos en isométrica.
Y porsupuesto, seguiremos demostrando que...
cualquiera puede hacer juegos
¿ Cualquiera ?
Ahora con funciones 3D
Ahora podrás crear juegos en red
Editor de mundos tridimensionales
Browsers para todo tipo de ficheros
Generador automático de sprites
Rutinas de inteligencia artificial
Mapeador de niveles para juegos 3D
Instalador profesional configurable
Más de 1.000 Bugs solucionados
Código optimizado para Pentium
Rutinas para manejo de textos
Sistema de sonido mejorado
Compilador más optimizado
Y un sin fin de funciones
DIV manía
POR FIN LLEGA DIV 2
AHORA CON OPCIONES 3D
Revista Oficial DIV GAMES
De venta en quioscos, grandes almacenes y tiendas especializadas
Teléfono distribuidores +34 91 304 06 22 (ext. 137)
sólo 4.995 ptas.
(bueno... habrá quien todavía no se aclare)
DIV 2 Manual de usuario
Manual de 348 págs.
LA HERRAMIENTA PERFECTA
Un nuevo entorno de desarrollo que ha evolucionado tanto en sencillez de uso como en potencia y capacidad. Y además se han incluido un gran numero de herramientas nuevas que harán que DIV 2 y tu imaginación formen un equipo perfecto.
COMPATIBILIDAD 100% DIV 1
Esta versión de DIV 2 mantiene compatibilidad al 100% con la versión anterior y además incluye un gran numero de mejoras y aspectos perfeccionados que hacen que disfrutes aun más de los juegos que desarrolles.
HAMMER Technologies
Alfonso Gómez 42, nave 112
28037 Madrid. España
Tel: (91) 3.04.06.22
Fax: (91) 3.04.17.97
Distribución en Argentina • Take Off Multimedia • Pueyrredon 495
Tel / Fax: (1704) 656 8506 • E-Mail: net2land@net2land.com

Lo bueno, si breve ...

Las claves a la hora de optimizar programas

Muchas veces pensamos que mientras más largo es nuestro código y mientras más recursos usa, más bueno es nuestro programa. Esto no es del todo cierto y es mejor poseer un programa optimizado para poder utilizarlo en tantas máquinas como sea posible.

La mayoría de programadores se sorprenden cuando ven el código de un programa y delante de él aparecen cientos y cientos de líneas, que forman en conjunto ese juego que tanto nos ha gustado. Pero si analizamos detenidamente el código es muy probable que encontremos funciones redundantes y procesos que podrían haberse realizado en la mitad de líneas. ¿Quién de vosotros no ha pensado alguna vez: "pues yo habría hecho este código así en lugar de como está aquí"? Pues bien, uno de los pasos que los programadores noveles no acostumbran a realizar, y realmente es muy importante a tener en cuenta, es la optimización del código, de los gráficos y del diseño general de nuestro juego.



Gran capacidad de imágenes conseguidas por un grupo que empieza.

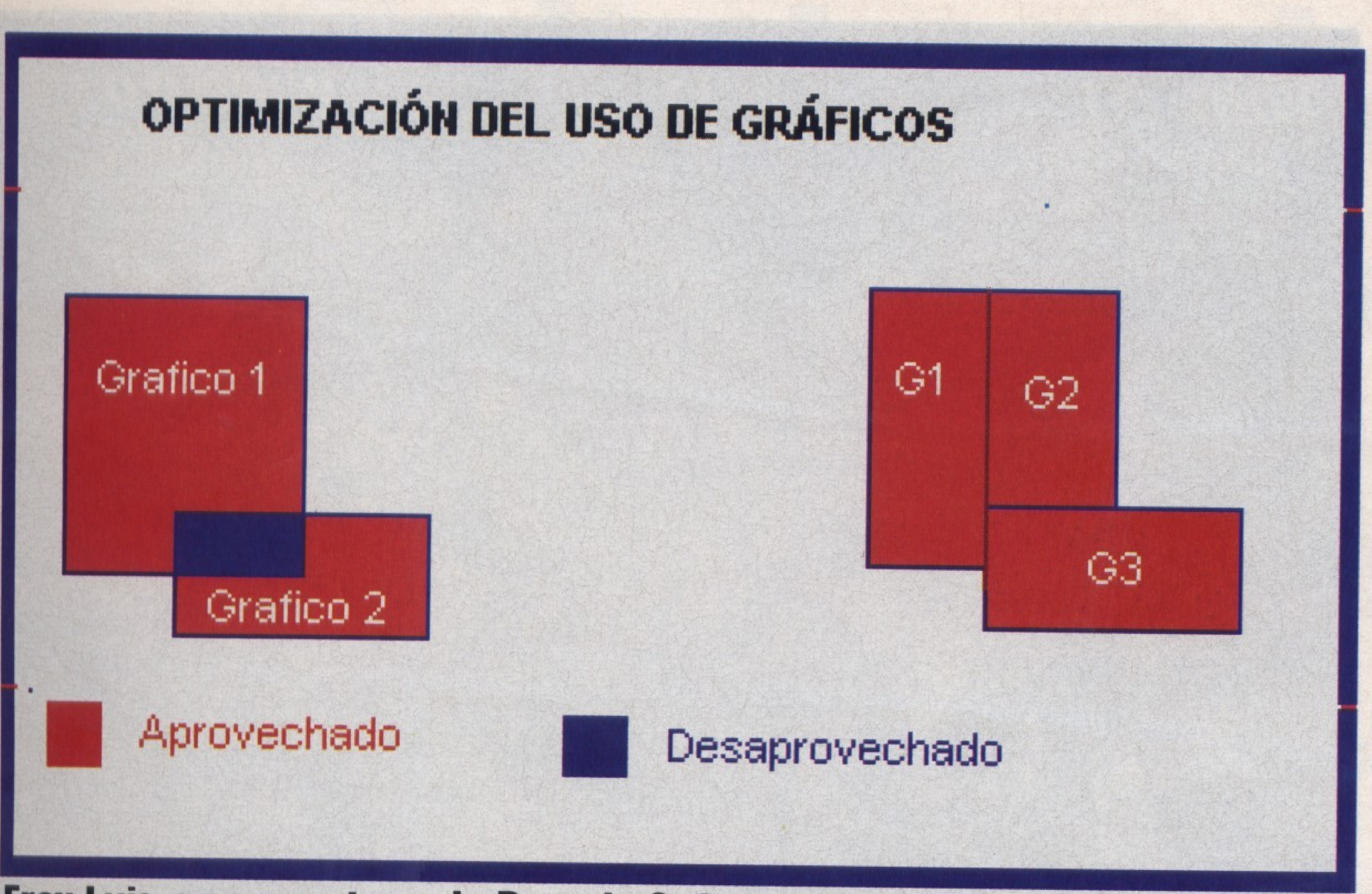
BCC (Breve, Claro, Conciso)

Breve, claro y conciso son las tres palabras que debemos tener en la cabeza a la hora de realizar nuestro programa. Son las reglas que debemos seguir para conseguir un programa rápido, que ocupe poca memoria y que realice pocos cálculos matemáticos.

Hay cuatro reglas básicas a tener en cuenta a la hora de optimizar nuestro código:

- *No optimices hasta acabar el programa.* Muchos programadores optimizan a medida que crean el programa. Esto es un error grave que a veces se paga con muchas horas de rehacer códigos y volver a crear gráficos, y dando como resultado programas ininteligibles e incorrectos. Sigue este consejo: escribe tu programa sin preocuparte de las posibles optimizaciones y concéntrate para estar seguro que el código es limpio, correcto e inteligible. Si es un programa muy largo o muy lento, cuando has acabado puedes considerar la posibilidad de optimizarlo.
- *La regla de 80/20* . En muchos casos el 80% de los resultados lo puedes obtener con el 20% del esfuerzo. A la hora de optimizar utiliza las herramientas de *debug* para encontrar donde se concentra el tiempo de espera para generar los resultados. Normalmente el 80% de los resultados se generan en un mismo momento, si tenemos claro esto podemos concentrarnos únicamente en esa parte del código para optimizarlo, con lo cual nos ahorraremos mucho esfuerzo.
- *Siempre prueba el código antes y después de los cambios.* Debemos tener claro cómo han afectado nuestros cambios al programa. Debemos testear las innovaciones que hemos realizado en cada ocasión. ¿Cómo si no sabremos que ha cambiado del programa con nuestras optimizaciones? Si nuestro programa únicamente se ha vuelto imperceptiblemente más pequeño o inapreciablemente más rápido, es mejor volver al código inicial, pues este resulta mucho más claro para realizar cambios. No siempre es mejor velocidad que claridad para después reparar *bugs* en la fáse de testeo.
- *Utiliza las funciones correctas y las estructuras correctas* ¿Para qué vamos a utilizar un tipo de dato *WORD*, cuando con un *BYTE* tenemos bastante? No utilices funciones lentas si puedes hacer lo mismo con algoritmos más rápidos. De la misma forma, para qué utilizar el método de ordenación de la burbuja cuando puedes utilizar el método *quicksort*, que es hasta 10 veces más rápido que el primero? Debéis tener en cuenta cómo realizáis vuestros programas: es importante utilizar estructuras rápidas antes que largos *arrays* de objetos, si podemos reutilizar gráficos rotando la paleta de colores, ¿por qué no hacerlo en lugar de crear gráficos nuevos?

Es importante utilizar un número de recursos tan pequeño como sea posible para que nuestro juego pueda ser utilizado en ordenadores más limitados sin que se nos caiga el alma al suelo mientras esperamos que aparezcan los gráficos en la pantalla, o mientras se calcula de qué manera afectará lo que hemos hecho a nuestro entorno. Un buen ejemplo de aprovechamiento de código lo podréis ver en el juego *Outcast*, que dentro de



Fray Luis, una aventura de Papada Soft en proceso de desarrollo.

una época donde las tarjetas 3DFX se han convertido en el pan nuestro de cada día, ha creado un entorno en tres dimensiones de gran belleza y rapidez con únicamente tecnología MMX. De esta manera se ha podido conseguir un buen juego, que funciona en un número mayor de equipos, debido a que necesita menos recursos para poder funcionar.

El mantenimiento

Ser capaz de entender un programa es importante en un lenguaje como DIV, donde el código puede ser utilizado tantas veces como deseemos y donde pueden ejecutarse varios procesos a la vez. Tener un código claro también es una optimización: mientras menos tiempo debamos pasar editando, compilando y viendo código, más tiempo tendremos para intentar que realmente el código funcione correctamente.

El mejor consejo que puedes seguir es crear un programa que sea inteligible por un humano y no por una máquina. Eso sí, debemos tener en cuenta el no pecar por el lado contrario: no es una buena optimización tener un proceso llamado: *proceso_que_llama_al_grafico_para_que_se_mueva()*, cuando podemos llamarlo *mueve_grafico()*.

De la misma manera no debemos caer en la idea errónea de pensar que todo el mundo sabe lo que nosotros sabemos. Normalmente trabajamos con otra gente y compartimos códigos para un mismo programa. Por ello, para las líneas en las que se realizan muchas funciones, debemos dejar claro en qué orden se ejecutarán: por ejemplo en las operaciones matemáticas deberíamos incluir los paréntesis necesarios, no todo el mundo conoce la prioridad de las operaciones y eso puede provocar muchas confusiones.

Otra cosa a tener en cuenta es la utilización de algoritmos revisados. Normalmente las funciones nuevas que incluye *DIV 2* pueden llegar a tener más *bugs* que las funciones de *DIV 1*, pues estas últimas ya han sido revisadas con anterioridad y han sido reparadas en la segunda versión. Es por ello que a veces lo antiguo es mejor que lo moderno, si no estamos seguros de que funciona todo correctamente.

Para acabar con este punto solamente decir que muchas veces es mejor utilizar funciones propias de *DIV* que implementar nuevas DLLs. A pesar de que *DIV* está preparado para ello siempre podemos encontrarnos con problemas de incompatibilidad entre ambos lenguajes, y nosotros podemos no darnos cuenta de ello.

El coste de ejecución

Muchos debéis pensar que todas las funciones y datos trabajan con la misma velocidad debido a que ya los tenemos en memoria cuando los ejecutamos. Esto es totalmente falso y es algo que asimilan los programadores al iniciar su aprendizaje: no es lo mismo incrementar un dato tipo *byte*, que incrementar un dato tipo *word* y de la misma manera ocupa menos memoria el primero que el segundo. Estos pequeños detalles son los que debemos tener en cuanta a la hora de optimizar un código por su "coste de ejecución".

No vamos a entrar en detalles sobre el coste de ejecución de cada función, pues eso podría ocuparnos número enteros de la revista, simplemente os vamos a dar algunos consejos a seguir para tener en cuenta si nuestro coste es demasiado elevado:

Funciones y datos no a la misma velocidad trabajan.

- *Mientras menos comparadores mejor.* El coste de comparar dos elementos es muy elevado, sobretodo en la sentencia *IF*. Debemos intentar utilizar funciones matemáticas básicas (suma, resta, división y multiplicación) que son mucho más rápidas y tienen un bajo coste. Por ejemplo el siguiente código, que seguramente habréis utilizado de una forma parecida en vuestros juegos:

```
if (camina==true) x++;end;
```



Gran calidad de imágenes conseguidas por un grupo que empieza.

Podría optimizarse de la siguiente forma:

```
x=x+camina;
```

Debemos recordar siempre que los *boleanos*, si están bien implementados internamente, devuelven un 0 si son falsos ó un 1 si son verdaderos.

- *Local mejor que Global.* Debéis tener en cuenta siempre lo siguiente, una variable global siempre está en memoria y realizar cambios en ella, como asignarle datos o realizar operaciones, es más lento que utilizar datos de tipo local. Para hacer una estimación aproximada de los costes de ejecución debéis saber que acceder a un tipo de dato global cuesta más o menos el doble que acceder a un dato de tipo local.
- *Estructuras claras.* El uso de estructuras es ya de por sí lento, pero realmente es mejor utilizar estructuras que muchas variables distintas. Pero ello no quiere decir que nuestras estructuras no deban estar declaradas de una forma óptima. Miremos el siguiente ejemplo que podría ser el de un mapa *tileado*:

```
STRUCT mapa
        mapax[12];
        mapay[12];
        graphic[144];
END;
```

Realmente es poco óptimo y podríamos utilizar un acceso más rápido utilizando un número menor de parámetros:

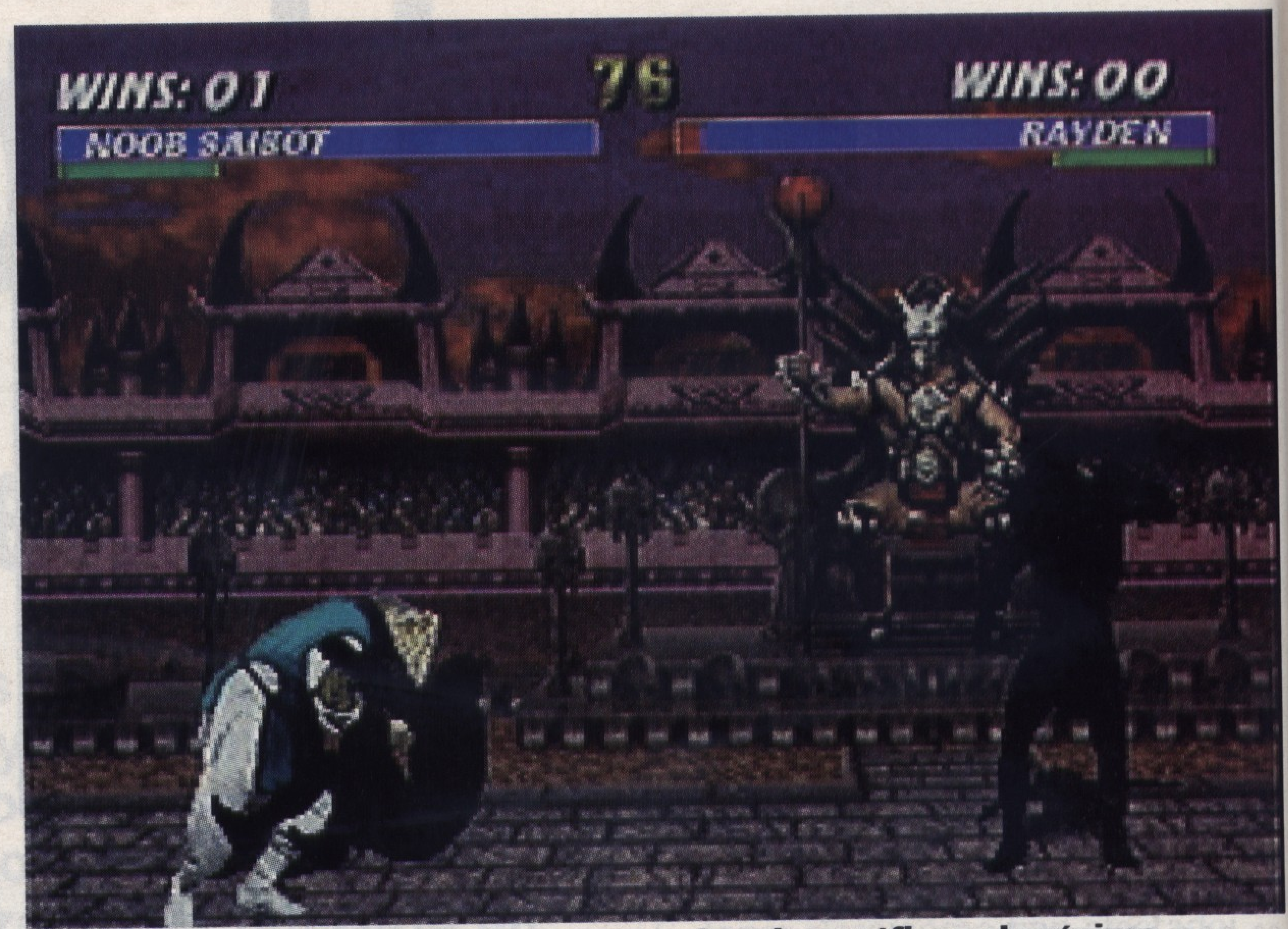
```
STRUCT mapax[12]
    STRUCT mapay[12]
        graphic;
    END;
END;
```

Las funciones matemáticas son muy rápidas y tienen un bajo coste.

Mientras menos accesos deba hacer nuestro programa para localizar una variable más rápido será de ejecutar y mostrar resultados.

- *La ley de Murphy.* En programación se cumple siempre esta ley: "un programa siempre va de la peor manera que puede ir". Es decir, para calcular su coste de ejecución debemos ponernos siempre en el peor de los casos, pues es el que se cumple la mayoría de las veces. Si un proceso es más largo pero sólo debemos ejecutarlo tres veces, en el peor de los casos, para obtener un resultado y, en cambio otro más corto, en el peor de los casos, debe ejecutarse treinta veces realmente es mucho mejor el primero de ellos a pesar de ser más largo.

Solamente debéis tener en cuenta una cosa, mientras mas simples sean las estructuras que utilizamos más fácil es que el programa calcule los resultados deseados.



MK Trilogy, un juego donde se aprovechan los gráficos al máximo.

Un código pequeño, un espacio pequeño

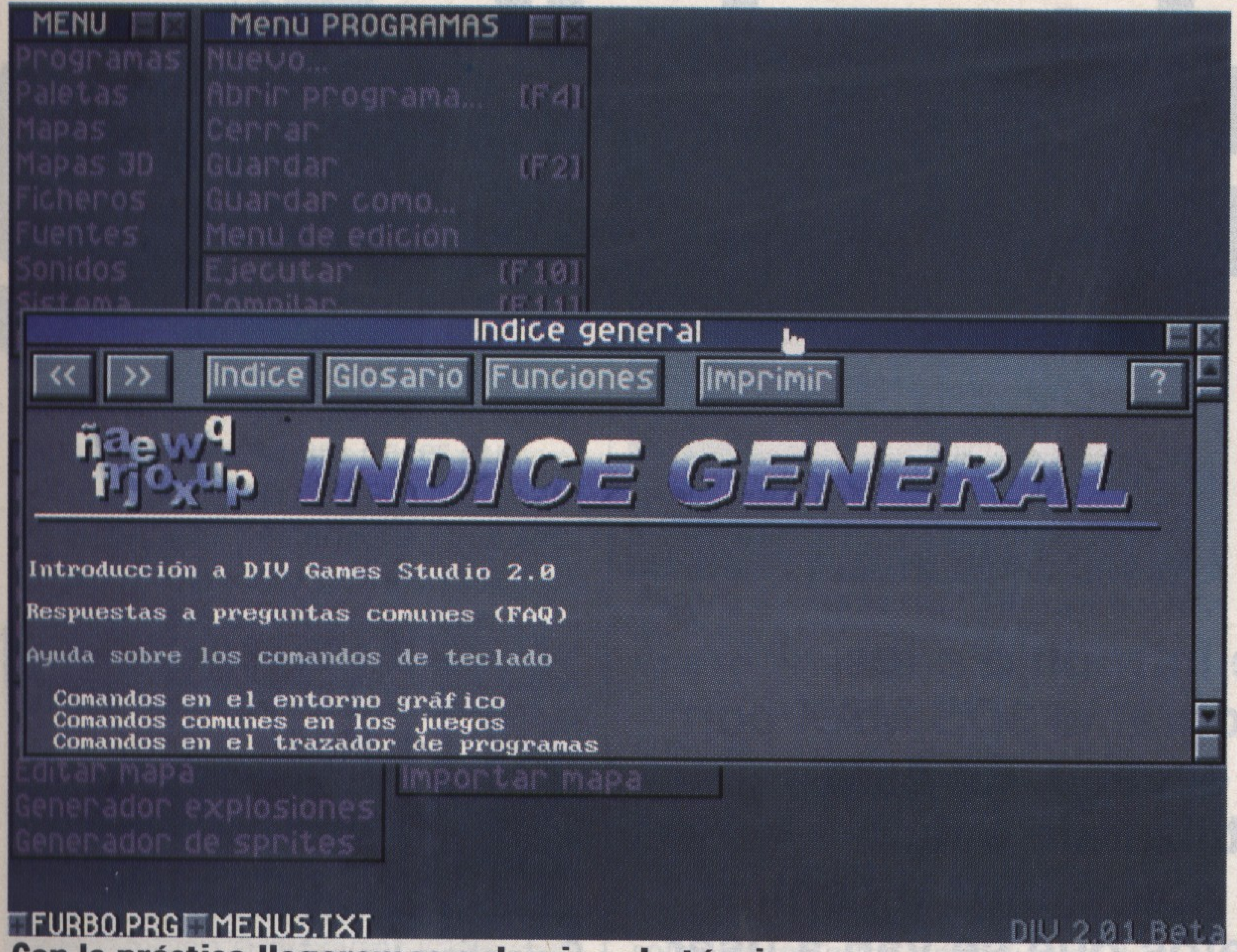
Debemos procurar que nuestro código no ocupe más líneas de las necesarias, pues esto repercute directamente en tres cosas: la cantidad de espacio necesario para almacenarlo, la cantidad de tiempo que tarda en compilarse el código y por último, mientras más código tengamos, más líneas deberemos revisar. Por ello debemos seguir una serie de pasos básicos para reducir la cantidad de líneas y de espacio que ocupa nuestro programa:

- *Cread procesos generales, antes que específicos.* Si es posible utilizar un mismo proceso para que pueda hacer mucho más trabajo, a pesar de que le vayamos a pasar mayor número de parámetros, debéis hacerlo. Por ejemplo, si podéis utilizar el mismo proceso que mueve al personaje para mover a los enemigos; hacedlo, pues esto hará que nuestro código sea mucho más rápido y más pequeño.
- *Reutilizad los gráficos tantas veces como os sea posible.* Habréis observado que en muchos juegos se utilizan gráficos de formas repetidas; únicamente cambiando los colores de los mismos mediante la rotación de la paleta. Tenemos un clarísimo ejemplo de este tipo de "morro" (pero que resulta bastante útil) en el título *Mortal Kombat Trilogy*, donde podéis comprobar cómo ejércitos de ninjas y de mujeres guerreras idénticas invaden este juego de lucha. Por ello, si nos es posible reutilizar gráficos de una forma agradable debemos hacerlo, pues hará que nuestro programa sea mucho más "ligero".
- *Utilizad ficheros con datos siempre que podáis.* Si debemos hacer grandes declaraciones en *arrays*, o debemos inicializar tablas con un número considerable de posiciones, es mejor hacerlo una única vez, guardar la información en un fichero y después acceder a ese fichero donde ya tenemos la inicialización hecha. No debemos tener miedo de guardar información en el disco, pues incluso *DIV 2*, nos permite encriptar datos para su propia seguridad.

Un código debe ser breve y claro para que se compile rápido y lo podamos comprender. Es bueno utilizar comentarios para tener las cosas claras, pero acordaros de quitarlos cuando compiléis el juego, ya que realmente no son necesarios.

La velocidad

Para acabar debemos procurar que nuestro código sea rápido: mientras menos tiempo necesitemos para obtener un resultado, mayor calidad tendrá nuestro juego. La utilización de los recursos estrictamente necesarios es el método de conseguir que un juego sea rápido y por lo tanto agradable para el usuario final. Se puede hablar largo y tendido



Con la práctica llegaremos a dominar la técnica.

sobre la velocidad de los juegos. Detrás de un programa perfecto, si es que existe, hay muchas y muchas horas de posprogramación. Por ello os vamos a dar algunos consejos de por dónde podéis empezar a optimizar.

- *Utilizad gráficos pequeños en lugar de superponer dos gráficos.* Si una parte de un gráfico va a estar siempre oculta tras otro gráfico y nunca se va a ver, ¿para qué entonces debemos tenerla en memoria?. Imaginemos el dibujo de una casa y el dibujo de un coche que está aparcado justo delante. El coche tapa parte de la casa y siempre está quieto: entonces, ¿ por qué no dividir la casa en dos gráficos pequeños y nos olvidamos de la parte que nunca se va a ver?.
- *El uso de la concatenación de* Strings. Parece realmente sencillo, ahora que *DIV 2* nos lo permite, el uso de las concatenaciones de *Strings* para la creación de nuestros códigos o para ahorrar líneas de nuestro programa. Pero debéis tener en cuenta que la concatenación es realmente un proceso muy lento internamente. Por ello a veces es mejor hacer asignaciones, a pesar de que ocupen mas líneas que el hecho de utilizar concatenaciones de *Strings* u otras operaciones similares.
- *A veces es mejor un código ligeramente complicado.* Debemos tener en cuenta que si necesitamos ganar velocidad a veces es necesario sacrificar un código sencillo e implementar programas más complejos. Por ejemplo, realizar operaciones con un *array* de dos dimensiones es mucho más costoso que utilizar un *array* de una dimensión, al cual podemos acceder mucho más fácilmente. Por ejemplo, en este caso, si nuestro *array* de dos dimensiones es de longitud *ndatos en mfilas*, podemos utilizar un *array* de una sola dimensión pero más grande, y acceder a los datos de la forma *dato=(linea_actual*ndatos)+dato_que_buscamos;*.

Para acabar sólo decir que para saber si un programa es veloz, sólo debéis tener en cuenta el número de recursos que consume, lo cual podréis comprobar mediante el mismo DEBUG del DIV, una herramienta muy útil que muy poca gente utiliza.

Trucos para optimizar en DIV 2

DIV 2, además de habernos proporcionado un nuevo motor y algunas funciones nuevas para el uso de *strings*, también nos ha proporcionado nuevas funciones para la optimización de código y la liberación de recursos, así como una serie de opciones de compilación que pueden ayudar a optimizar nuestro programa. Además de éstas también contamos con opciones muy útiles, que ya aparecieron en *DIV 1*, como son las señales.

Hablando de éstas últimas los usuarios de *DIV* deben tener en cuenta que, mientras un proceso está dormido o congelado, se reduce notablemente su coste en el programa. Esto es debido a que no tiene que estar continuamente ejecutándose. Es por ello que el uso de los *signal: s_sleep,s_freeze* y *s_wakeup*, debería utilizarse sólo cuando sabemos que no es necesario que se estén ejecutando códigos continuamente. Por último también debemos tener en cuenta que hay que evitar que los procesos se ejecuten continuamente si no los necesitamos más. Siempre que un proceso deje de ser útil debemos acabar con él, ya sea mediante programación, o con el uso de la señal *s_kill*.

También debemos procurar tener cargado solamente lo que necesitamos a cada momento. Si cargamos ficheros, paletas, fuentes, o lo que sea, debemos acordarnos de liberar la memoria utilizada mediante el *unload pertinetne*. De esta manera el ordenador volverá a disponer de los recursos que ocupaban estos ficheros.

Entre las nuevas funciones que encontramos en *DIV 2*, podemos hallar tres realmente útiles para una gestión dinámica, y por tanto optimizable, de la memoria: *free()*, *malloc()*, y *memory_free()*. Mediante estas funciones podemos reservar posiciones de memoria para nuestros tipos de datos, pudiendo controlar el espacio ocupado, y podemos liberar los recursos cuando nosotros deseemos. El uso de estas funciones es complicado y os aconsejo que experimentéis un poco con ellas antes de usarlas en vuestros programas. Pero el dominio de estas funciones, junto con las otras opciones que tenemos de liberar recursos, así como de guardar información en disco para una mejor gestión de lo que necesitamos, es lo que va a hacer que vuestros juegos sean realmente óptimos en *DIV*.

Cuando un proceso deje de ser útil debemos eliminarlo

Para acabar

Terminamos simplemente recomendando que probéis vuestros algoritmos y experimentéis con ellos. No os conforméis con el primero que funcione pues, normalmente, nunca es el mejor. Además podéis consultar algunos libros que pueden ayudaros a optimizar vuestros programas, que a pesar de estar en inglés, son realmente muy interesantes:

Writing Solid Code de Steve Maguire

Code Complete de Steve McConnell.

Una buena lectura a tiempo puede ahorraros muchos esfuerzos en el futuro.

Jordi Martín

Introducción al diseño gr

Modelando una nave espacial

En el artículo anterior se introdujo a los usuarios de DIV en el programa 3DS MAX con el que ya se generó una escena. En este artículo se va a seguir utilizando este programa, pero esta vez nos vamos a centrar más en el diseño de objetos que se van a usar en el juego mismo y no en la presentación.

En este artículo se va a introducir un apartado nuevo llamado "dudas de los lectores" y en el caso de que los lectores vayan mandando sus propias imágenes, se irán publicando en esta sección o en el CD. Es decir los usuarios que quieran colaborar pueden hacerlo mandando sus imágenes o dudas a dmc@mundomail.net

Un juego de naves

En este artículo se va a empezar a generar todo lo necesario para hacer un juego de naves, es decir, las mismas naves, los asteroides, la pantalla de presentación, los marcadores, etc. Para ello se van a utilizar los programas *3DS MAX* y el *Adobe Photoshop* (se pueden seguir los mismos pasos con algún programa que no sea *Photoshop* aunque algunas cosas cambian). Al igual que en la creación de la copa del artículo anterior, primero realizaremos el modelado de la escena, después pasaremos a ponerle textura. Y finalmente, antes de hacer el *render*, deberemos iluminar y enfocar la escena de la manera apropiada. En este número se va a tratar más a fondo los parámetros de la opción *render*.

Se debe tener una idea clara de cómo debe ser la nave

Modelando la nave

A la hora de modelar un objeto se debe pensar cuál es la manera más fácil y más apropiada de hacerlo. El autor de estas palabras se ha decantado por hacer la nave a partir de un cubo, ya que no requiere la utilización de técnicas complejas y puede llegar a dar unos resultados estupendos.

Lo primero que se debe hacer es crear un cubo (se explicó como crearlo en el primer artículo por lo que no se insistirá mucho en la técnica). Para crearlo se debe ir al panel *create* y entrar en el apartado de *geometry*. Dentro de dichos apartados se debe seleccionar del *listbox* la opción de *primitives*. Una vez realizado esto se debo hacer clic sobre el botón que pone *box* y a continuación ya se puede crear desde cualquier vista. El resultado de la operación realizada debería ser parecido al de la figura 1.

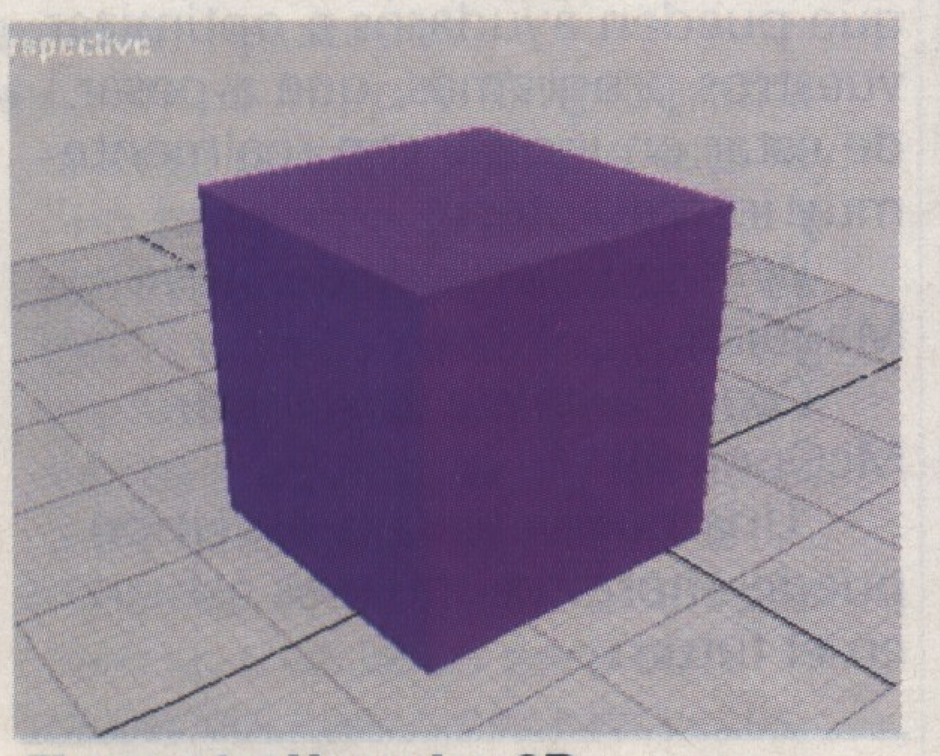


Figura 1 - Un cubo 3D

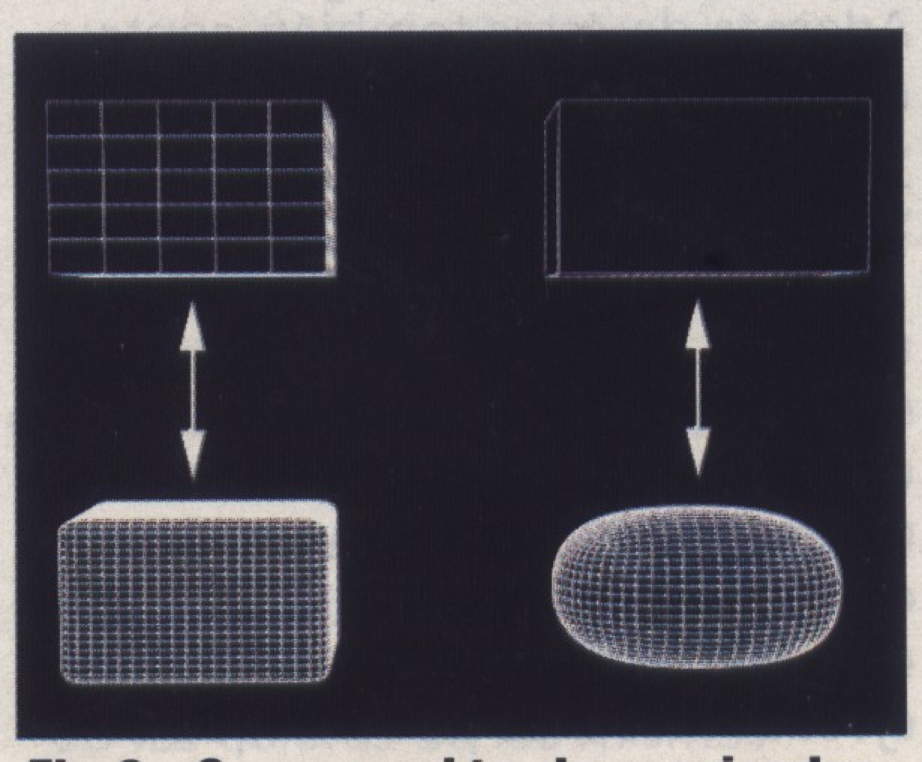


Fig.2 - Comparación de suavizados.

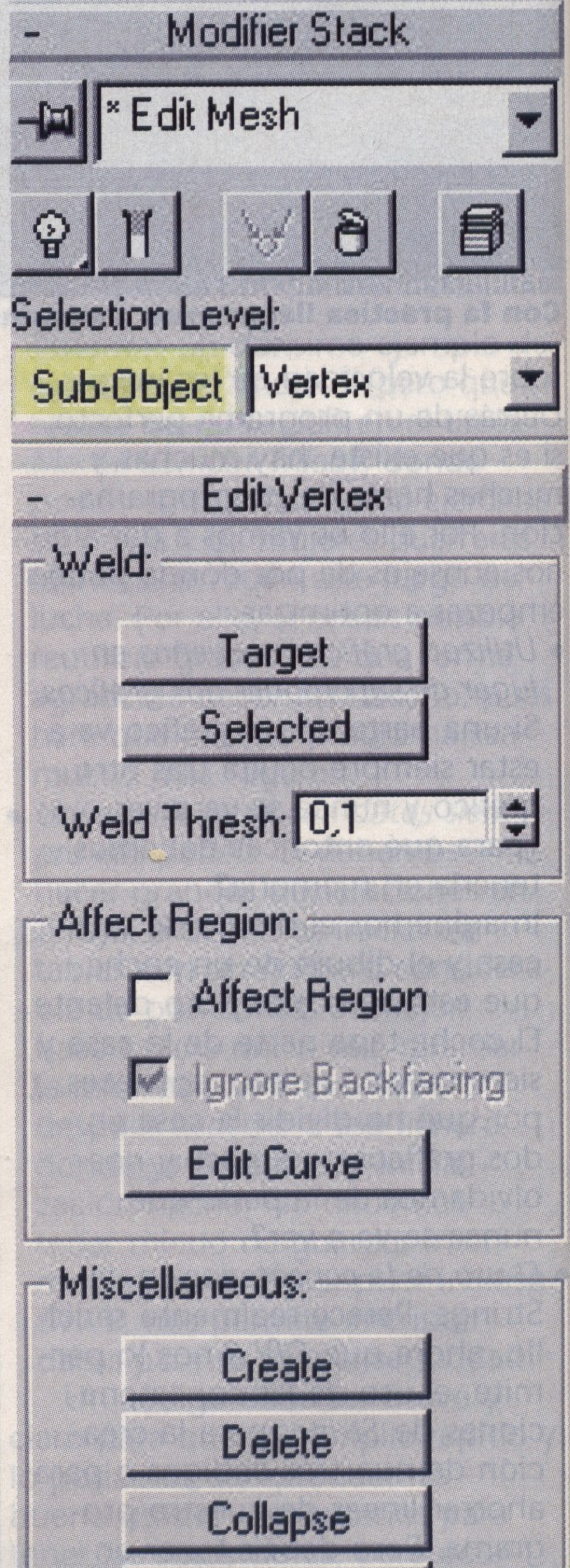


Figura 3 - Modificador "Edit Mesh".

La técnica que se va a utilizar para modelar la nave es *extruir* caras para coger una forma aproximada y después aplicar el modificador *meshsmooth* que provoca un suavizado del objeto. Cuantas menos caras tiene el objeto más suavizado

gráfico

queda, pero el modificador también tiene parámetros para hacer que afecte más o menos el suavizado.

El número de caras de un cubo viene determinado por los parámetros que se le insertan al crearlo. Sin embargo, si no le hemos aplicado ningún modificador, se le puede cambiar el número de caras desde el panel *modificar*. En la figura 2 se puede observar la diferencia entre un cubo con pocas caras suavizado y un cubo con muchas caras también suavizado.

Ahora se debe proceder a modelar la nave. Lo primero de todo, se debe tener una idea clara de cómo debe ser la nave (su forma exterior). Éste es un concepto muy importante a tener en cuenta en el mundo de las 3 dimensiones. El autor recomienda hacer unos 3 dibujos (desde arriba, de frente y de perfil). El dibujo en 3 dimensiones no hace falta hacerlo puesto que para ver nuestra nave u objeto resultante en 3 dimensiones ya utilizamos nuestro programa.

Los dibujos no deben estar trabajados excesivamente, solo deben ser lo suficientemente claros para que la persona que vaya a modelar la nave sepa cómo es, es decir, hay que dar una idea global de cómo

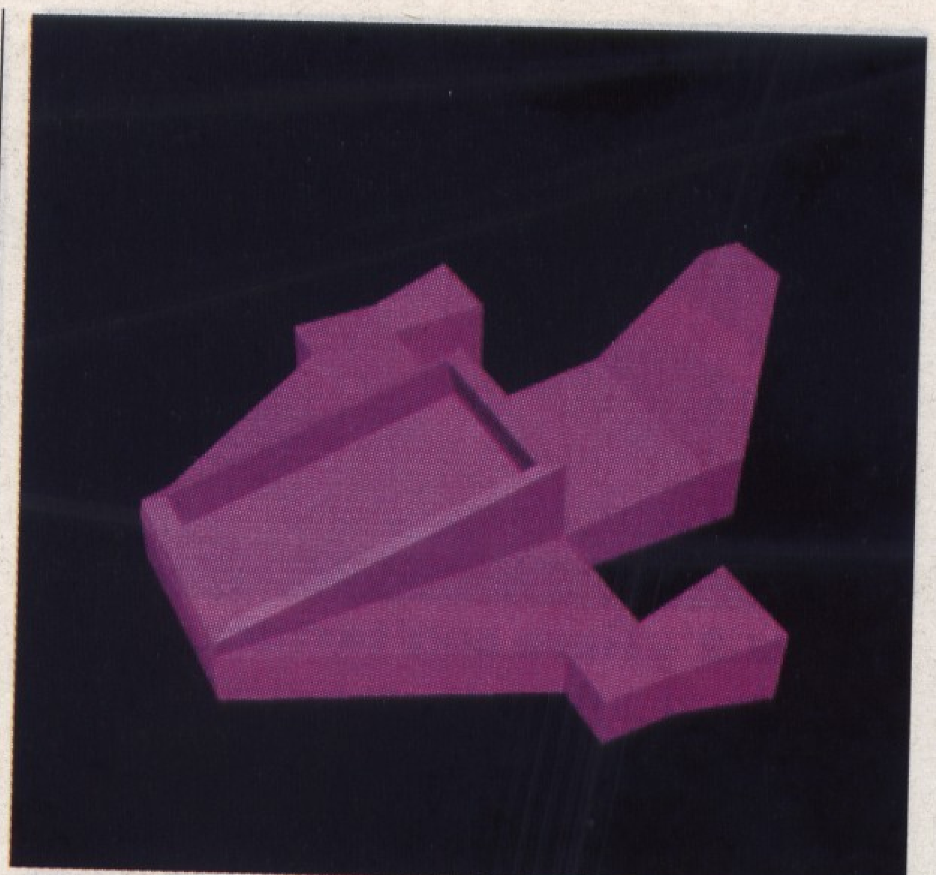


Figura 4 - Nave sin "Meshsmooth".

va a ser el objeto en sí.

Ahora ya se puede empezar a crear la nave. Para ello se debe ir al panel de modificadores y aplicar el modificador *edit mesh*. Sin tocar ninguna de las opciones del modificador los resultados deberían ser semejantes a los de la figura 3.

Lo que se debe hacer ahora es cambiar el contenido del *listbox*, donde hay escrito *vertex* debemos ponerlo en *face*. De esta manera ahora lo que se va a modificar va a ser la cara, que previamente deberá ser seleccionada. Una vez que estamos en modo *face*, se debe ir a cualquier vista (la que vaya mejor en cada caso) y pulsar sobre la cara que se quiera modificar. Esta cara se mostrará de color rojo y entonces ya se podrá modificar.

El siguiente paso que se debe dar es extruir la cara. Para ello se debe mirar en las opciones del modificador y fijarse especialmente en un apartado llamado *extrusion*. En él hay una opción llamada *amount*. Esta opción sirve (cuando se tiene una cara seleccionada) para extruir la cara. Para ello debemos hacer clic y, sin dejar de apretar el botón, mover el ratón hacia arriba o hacia abajo para cambiar el valor de la extrusión. Una vez hecho esto se deben seleccionar las caras que se prefieran para ir haciendo una nave. De momento va a quedar algo cuadrada, pero después de aplicarle el modificador *meshsmooth* quedará muy suavizada. En la figura 4 se puede observar una nave realizada por el autor sin el modificador *meshsmooth*. La



Figura 5 - Nave con "Meshsmooth".

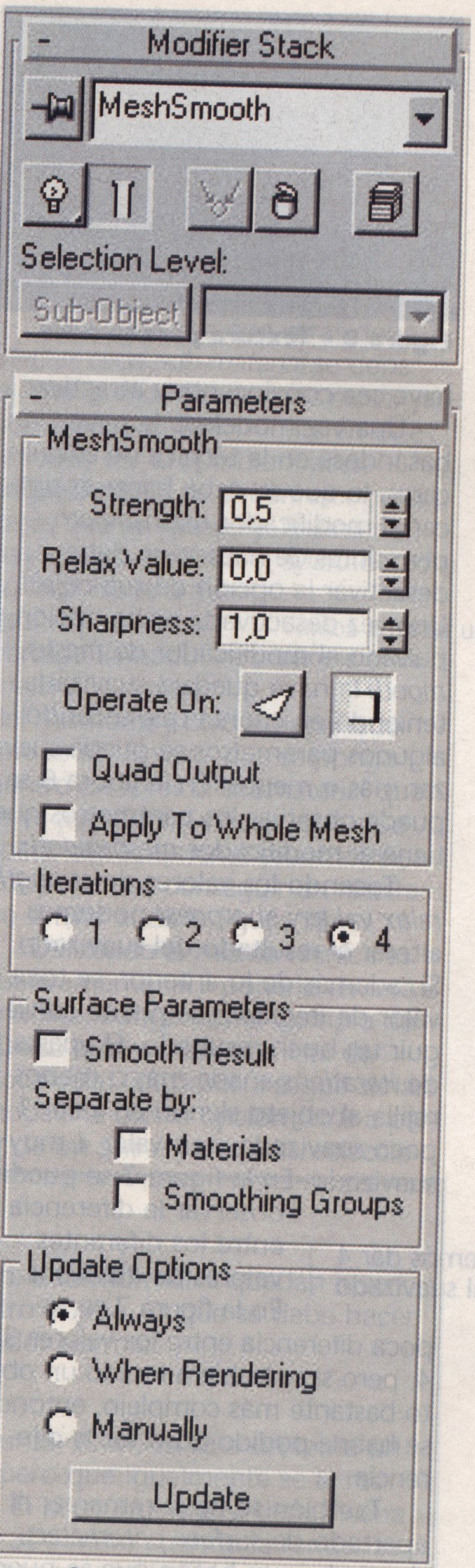


Figura 6 - Modificador "Meshsmooth".

imagen 5 es la misma nave pero con el modificador aplicado.

Las imágenes 4 y 5 muestran la nave que ha modelado el autor, pero se pueden conseguir resultados mucho mejores empleando esta técnica. Lo que se debe hacer es trabajárselo mucho para que la

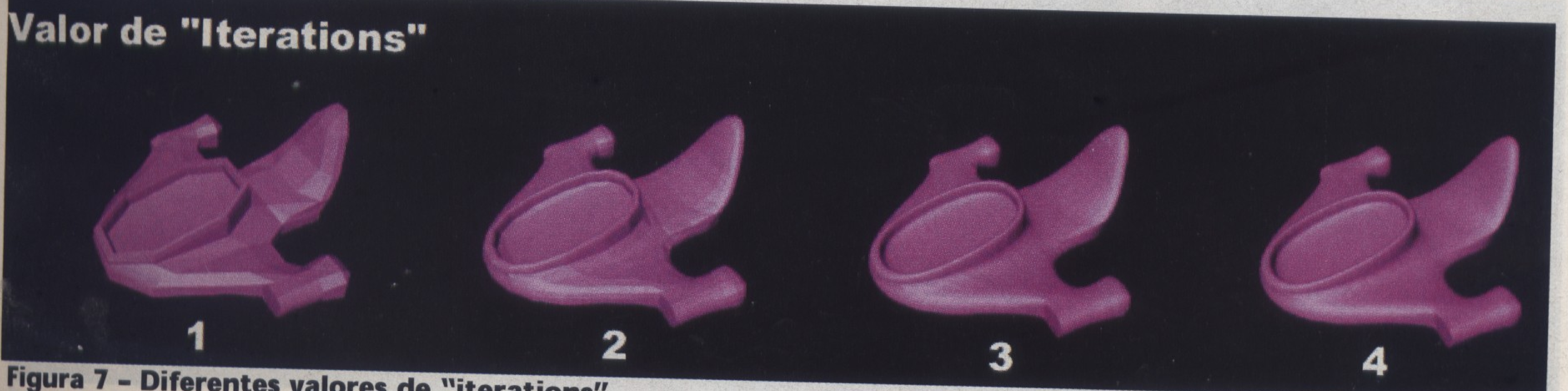


Figura 7 - Diferentes valores de "iterations".



Figura 8 - Textura para la nave.



Figura 9 - Nave con textura.

nave sea convincente y atractiva.

Una vez modelada la nave, basándose en la técnica de extruir caras, lo que se debe hacer es aplicar el modificador *meshsmooth,* pero antes de hacerlo se debe desactivar la opción de *sub-object.* Una vez desactivada dicha opción y pulsado el modificador de *meshsmooth* la nave quedará suavizada, teniendo en cuenta que tocando algunos parámetros se puede suavizar más o menos. En la figura 6 se puede observar los parámetros que tiene el modificador *meshsmooth.*

Tocando los valores de *strength, relax value* y *sharpness* podemos alterar el resultado del suavizado. Si, además de lo anterior, se toca el valor de *iterations* se puede conseguir un buen resultado. El apartado de *iterations* añade más o menos rejilla al objeto siendo en valor 1 poco suavizado y en valor 4 muy suavizado. En la figura 7 se puede observar la diferencia entre los diferentes valores de *iterations.*

Podemos dar 4 valores al suavizado

En la figura 7 se ve poca diferencia entre los valores 3 y 4, pero si se hubiera tenido un objeto bastante más complejo, entonces se habría podido observar la diferencia.

También se debe remarcar el apartado de *surface parameters,* donde hay un botón que se puede activar llamado *smooth result.* Lo que se logra con ello es que, en la representación de la escena (render), no se vean cuadrados de suavizado, sino que parezca una superficie totalmente lisa y orgánica al mismo tiempo. Para la creación de la textura podemos usar un *bitmap (mapa de bits)* cualquiera, que colocaremos en las rejillas de *diffuse* y *bump.* Se explicó su posición en el artículo anterior pero, por si no se recuerda, estas rejillas están en la opción *MAPS* del editor de materiales.

En el ejemplo del artículo se ha aplicado una textura como la de la figura 8, pero se puede utilizar cualquier tipo de textura.

Para la rejilla *diffuse* se utilizó una imagen como la de la figura 8 pero para la rejilla de *bump* se debe utilizar otro tipo de textura. Generalmente en el *bump* se utilizan imágenes en blanco y negro para resaltar el relieve que se va a provocar. Para hacer el *bump* de la nave sería interesante utilizar un *bump* con cuadrados y líneas cruzando, como si se tratara de pequeños compartimentos o una especie de salidas de aire de la nave. En la figura 9 se puede observar un *render* de la nave con una textura en *diffuse* pero con ninguna textura en *bump.*

Teniendo en cuenta que la nave modelada va a ser utilizada en un juego, por lo general, es típico hacer la representación de la misma (render) desde arriba y después inclinándose hacia los lados, de manera que en el juego coja un aspecto más tridimensional.

Para realizar los *renders,* mencionados anteriormente, lo que se debe hacer es lo siguiente: primero realizar un *render* desde la vista *top*; a continuación lo que se va a hacer es rotar el objeto (no la cámara). Esto se hace de la siguiente manera: si uno se fija en la barra que hay en la parte de arriba del *3DS MAX* se puede comprobar que existen los botones mover, rotar y escalar.

También hay otros 3 botones que son los ejes en los que se va a mover, rotar o escalar. Si tenemos un objeto seleccionado y se pulsa en los botones anteriormente mencionados, es decir, rotar en nuestro caso, y escogemos el eje correcto, ya estamos preparados para mover la nave. El resultado de esta operación, después de haber generado los 3 *renders,* sería algo parecido a la figura 10. Hay que tener en cuenta que la figura 10 es una composición, tu tendrás las imágenes por separado.

Para darle un poco de vida al juego se recomienda darle una iluminación adecuada.

Otra cosa a tener en cuenta a la hora de generar un objeto es para qué se utilizará. Por ejemplo, la nave que se ha modelado es para meterla dentro del juego por lo que el valor de *iterations* no tiene porque ser muy alto. Pero si la nave se fuera a utilizar para una animación de presentación, o para una pantalla en la que se muestra información, entonces sería recomendable tener el valor de *iterations* de 4, corregir errores en la textura, etc.

Siguiendo con el juego, ahora se debería generar la nave o naves de los malos, pero eso se deja de la mano del lector, puesto que los pasos a seguir son los mismos que hemos visto hasta este momento. Es decir, *extruir* caras y hacer *meshsmooth,* pero hay un detalle que no se ha comentado. También se pueden generar los objetos a partir de primitivas, y a veces una buena textura puede darle más vida que una forma orgánica generada con *meshsmooth.* A continuación se va a mostrar un ejemplo en el que se modela una parte de una nave y se le da una textura que le da bastante apariencia de realidad.

La técnica de poner textura que se va a explicar a continuación es algo compleja. Si alguien se pierde y quiere que profundice más sólo tiene que enviarme un *e-mail* y su duda será respondida, tanto por correo electrónico como por la



Figura 10 - Render de la nave.

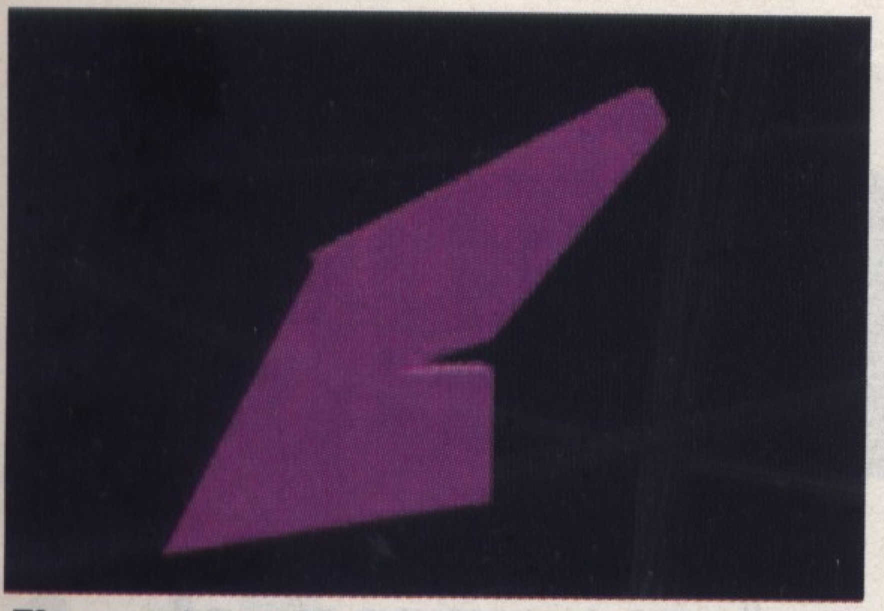


Figura 11 - Alerón sin textura.

revista, para que a los otros lectores que tengan la misma duda les quede resuelta.

Para empezar se debe crear un *spline* y extruirlo. El autor se ha decantado por hacer lo mismo que se puede ver en la figura 11 y ponerlo encima de la nave, como si de un alerón se tratara. Esto sólo es un ejemplo, se puede hacer la nave como uno quiera.

A continuación se debe hacer un *render* desde la vista *left*, de manera que veamos todo el cuerpo del alerón donde nos gustaría poner un dibujo o alguna otra cosa. Una vez que se tenga la imagen *renderizada* desde la vista izquierda, nos debemos ir a algún paquete de dibujo y, desde éste, recortar la imagen de manera que los bordes del objeto queden al limite del espacio de la imagen. Este último punto, el de recortar la imagen, es muy importante, puesto que si no se realiza adecuadamente puede provocar que desde el programa 3D no tengamos el resultado deseado.

Una vez pintada la zona por donde se supone que pasa el alerón y pintando también un poco los bordes se debe volver a *MAX* para aplicar la textura. Para hacerlo debemos entrar al editor de materiales y donde pone que ese material es *standard* cambiarlo por *multi-subobject*, una vez dentro se debe especificar cuantos *sub-materiales* tiene este material que se va a crear. Para cambiar el número de *sub-materiales* se debe pulsar en el botón llamado *set number* . En el caso de estar creando la imagen de este artículo sólo hace falta poner 2 ó 3. Ahora, si se intenta acceder a alguno de estos *sub-materiales,* se podrá observar que son como un material normal por lo que hemos dividido un material en varios *sub-materiales.* Cada uno de estos *sub-materiales* podrá ser aplicado a una cara diferente del objeto (no necesariamente a una cara, puede estar aplicado a muchas caras).

Siguiendo con la textura del alerón lo que se debe hacer es entrar en uno de los *sub-materiales* y en el apartado de *MAPs* insertar el *bitmap* creado en la rejilla *diffuse.* Una vez realizado esto se debe aplicar el material al objeto (en el caso del ejemplo del artículo al alerón).

Si ahora no se toca nada, el *3DS MAX* habrá asignado cada *sub-material* a una cara cualquiera del objeto, pero esto se puede corregir fácilmente.

Para asignar a que caras debe afectar cada *sub-material* se debe utilizar el modificador *edit mesh,* ya comentado anteriormente en este artículo, e ir al apartado *faces.* Seguidamente se deben escoger las caras que se deseen y, finalmente, hay que meterse en el apartado *edit surface.* Dentro de éste hay un subapartado llamado *material* y dentro del mismo hay otro subapartado que es *ID.* El valor que se ponga dentro de *ID* es el submaterial que se va a asignar. Es decir, que si se aplica correctamente este modificador podemos conseguir un resultado similar al de la figura 13.

Vamos a hacer un alerón para la nave

Utilizando la misma técnica es relativamente fácil ponerle textura a una lata. Pongamos el ejemplo de que se quiere crear una lata de Coca-cola. Los pasos a seguir serían modelarla con la opción *lathe* a partir de un *spline* y después, seleccio-

Dudas de los lectores

En esta primera entrega de "dudas de los lectores" tenemos 3 cuestiones de ARM muy interesantes y que pueden ayudar a muchos de los interesados en estos temas.

Primera pregunta:

Su primera duda se basaba en la creación de la copa del articulo número 3. Primero me dijo que, para que le saliera la copa entera, no debía dibujar el perfil sino toda la copa.

Respuesta:

La respuesta a esto es que en *lathe* debía fijarse bien en las opciones de *align,* donde hay 3 botones que son *min, max* y *center.* Se deben pulsar buscando el efecto esperado.

Segunda pregunta:

Su segunda duda seguía siendo sobre la copa. Ahora la había conseguido hacer con el medio perfil pero después se veía de una forma rara.

Respuesta:

A todos los que les haya pasado el mismo caso el autor de este artículo les ruega que no se alarmen porque es una cosa muy normal. Al hacer el *lathe* hay veces que las "normales" están invertidas. Las "normales" son la forma de ver un objeto, por decirlo de alguna manera, es decir, se puede ver la parte de dentro o la parte de fuera. Para corregir este error la opción *lathe* tiene un *checkbox* que se llama *flip normals,* y sirve para invertir las normales. Si no se encuentra esta opción, o las normales quedan invertidas al crear algún otro objeto, lo que se debe hacer es recurrir al modificador normal.

Tercera pregunta:

En la tercera pregunta, ARM, se va a un tema un poco más avanzado, pero que igualmente se le responderá en la revista. Él pregunta que si tiene hecha una animación desde *3DS MAX,* cuando la renderiza, cómo la puede guardar como animación.

Respuesta:

Bueno, la respuesta a esta pregunta también es fácil. Lo que se debe hacer es entrar en el apartado de *render,* y en él encontramos un apartado donde se escribe el fichero de salida. Hay que poner un fichero de extensión *AVI* o *FLC.* Además de eso, arriba del todo, en la ventana de *render,* por defecto pone *single frame* (un solo fotograma). Esto debe cambiarse a *range,* y ponerlo de 0 a 100.

Otra cosa a tener en cuenta en las opciones de *render* es que se puede cambiar el tamaño de la imagen resultante. Si nos fijamos por defecto está a 640x480 pero se puede cambiar a 320x200, o incluso introducir nuestros propios valores de manera que se podría generar un *render* de 30x2000 pixeles.

Cuando se ponen manualmente los valores del tamaño del *render,* es decir, si supuestamente hiciéramos ese *render* de 30x2000 pixeles no aplastaríamos la imagen sino que solo se *renderizaría* esa pequeña porción que entra en ese pequeño espacio.



Figura 12 - Alerón con textura.

nando las caras, ponerle la textura.

Ahora se mostrará un resumen con las cosas más importantes a recordar del modelado a base de extruir caras y *meshsmooth*:

- Cuantas menos caras haya mas suavizado quedará el objeto.
- Cuanto más alto es el valor de *iterations* del modificador *meshsmooth* más malla se generará con el objeto.

También es importante recordar que no siempre ésta es la mejor técnica para modelar naves o objetos (extruir caras y *meshsmooth*) ya que utilizar otras opciones, como el modelado con primitivas, da unos resultados que a veces son espectaculares.

Los asteroides

Siguiendo con el juego de naves ahora se debería crear un asteroide para ponerle más dificultad al juego. Modelar y poner textura a un asteroide es una de las cosas mas fáciles que se pueden realizar con *3DS MAX*. ¡Gracias a los modificadores crear un meteorito es un juego de niños!

Lo primero que se debe hacer es crear una esfera, para ello se deben seguir las mismas instrucciones que para crear un cubo con la única diferencia que al final se debe escribir *sphere*. Después de realizarla debería quedar algo parecido a la figura 13.

La esfera en sí es algo vulgar pero, si le aplicamos el modificador *noise*, podemos darle un aspecto arrugado y curioso. Si a eso le añadimos una buena textura de tierra podemos acabar teniendo un meteorito de aspecto muy atractivo.

El modificador *noise* se basa en distorsionar el objeto. El autor de este artículo recomienda bajar el valor del apartado *scale* e ir cambiando los valores de X, Y, y Z, de manera que quede arrugado. Una vez que el meteorito resultante guste al lector es la hora de aplicarle la textura. Para ello se deberá buscar una imagen con tierra, y aplicarla de una manera normal, es decir, sin *multi-sub-objetos* ni cosas parecidas.

Se pueden encontrar muchas texturas en CDs de revistas o incluso en Internet. Si se entra en http://www.3dcafe.com y después en *freestuff* se podrá encontrar un apartado entero sobre texturas y dónde conseguirlas. Pero se tiene que tener en cuenta que las texturas no son mas que imágenes inventadas o fotos escaneadas, por lo que si hay que generar una escena muy especial con unos materiales muy específicos lo que se debe hacer es hacer fotografías y escanear éstas para disponer de unas buenas texturas.

Para seguir con el juego de naves, actualmente una de las pocas cosas que nos quedan por hacer son las estrellas, pero éstas se pueden crear desde cualquier tipo de programa de dibujo o retoque fotográfico, como *Photoshop* o similares. Después habría que hacer un *scroll* para que vayan pasando.

En este artículo no se ha generado un personaje ni se ha empezado a hacer un juego de plataformas como se dijo en el artículo pasado, pero estas técnicas son totalmente aplicables a la generación de personajes y decorados. En el próximo número se va a seguir y acabar el juego de naves y nos centraremos en el tema de los decorados de los juegos. Este capítulo servirá para realizar desde los decorados de juegos de plataformas hasta los decorados de las pantallas de las aventuras gráficas. Pero todo esto será después de acabar el juego de naves (con código incluido que se va a incluir en el artículo siguiente).

Avanzando como será el juego que se a generar, solo se dirá que, aparte de tener una presentación 3D, tendrá varias cosas más, como marcador de puntos, música y buenos gráficos. Para poder compilar el código que se generará en el próximo artículo se deberá disponer de *DIV 2* o de la Beta, que ya salió publicada en esta misma revista.

Reiteramos que, al igual que en los artículos anteriores, si alguien tiene alguna duda, sugerencia, imagen, o lo que sea, que quiera compartir con el resto de los lectores que lo envíe a dmc@mundo-mail.net y será publicada en la revista o en el CD de ésta.

David Martínez

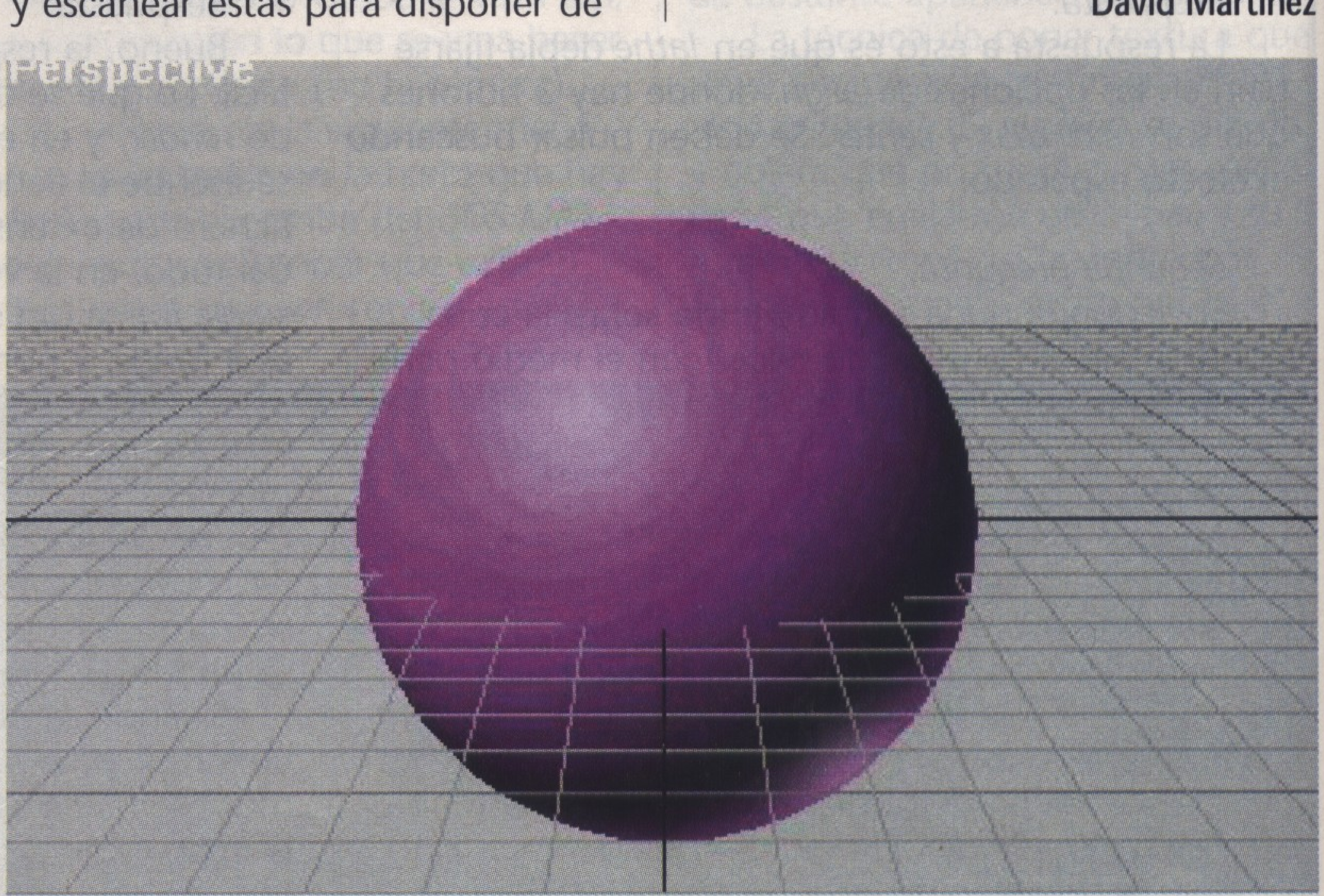


Figura 13 - Esfera normal.

Algoritmo objeto

Quiero eso en mi bolsa

En el número anterior ya empezamos a darle forma a nuestro juego. Conseguimos que, de forma rudimentaria, nuestro jugador pudiese realizar acciones sobre una diana de ejemplo. Incluso conseguimos que el resultado de una misma acción fuese diferente según la sucesión de acciones que se hubiese ejecutando antes. Sin embargo, esto sólo ha sido el principio.

Aún nos queda por definir cómo se van a realizar acciones entre objetos y, por norma general, nada es aparentemente útil donde se encuentra, debemos permitir a nuestro jugador acarrear objetos de una a otra sala del juego.

Definiendo

En primer lugar debemos tener una "bolsa". Esta bolsa nos va a permitir portar objetos. Podremos añadir y/o eliminar objetos de la bolsa, también debemos poder recorrer la bolsa. Para esto usaremos una estructura tipo lista que nos permitirá:

1. Añadir un objeto al final de la lista.
2. Eliminar cualquier objeto.
3. Recorrer la lista.

En esta lista almacenaremos los *ID* de los procesos correspondientes a los objetos que se encuentren en la bolsa. Debemos saber en todo momento el número máximo de objetos que tenemos en la bolsa, cuál es el último objeto en la bolsa y cuál es el actual. Para esto



En este castillo podemos empezar a equiparnos.



Ejemplo de uso de objeto.

usaremos la siguiente estructura:

```
bolsa_max   = 35;
STRUCT Bolsa
   cosa[35];
   ultima;
   actual;
   END;
```

- *bolsa_max*: indica el número máximo de objetos que podemos tener en la bolsa simultáneamente.
- *cosa[]*: es un *array* de *ID* de procesos que será controlado por funciones, dándole capacidad de lista. El número máximo de elementos en el *array* debe ser igual a *bolsa_max*.
- *Ultima*: es el último objeto en la lista (*array*).
- *Actual*: es el objeto actual en la lista (*array*).

Es muy importante no manipular directamente esta variable, dejándolo que lo hagan las funciones:

- *PROCESS bolsa_inicia()*: Inicia la variable "bolsa" con los valores adecuados. Dibuja los botones bolsa arriba y bolsa abajo. Redibuja la bolsa.



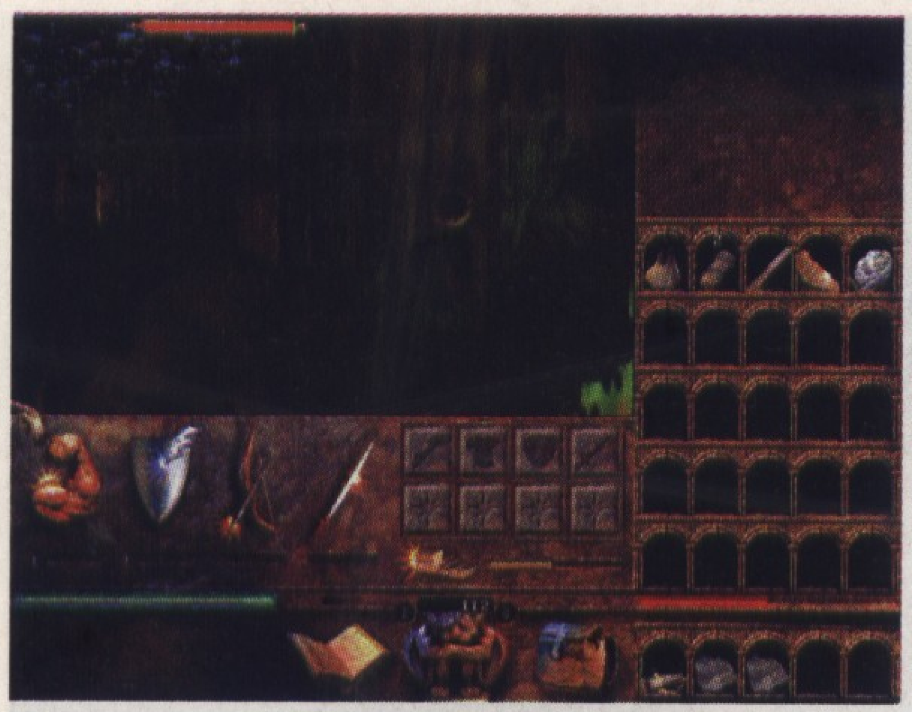
Debemos tener a mano nuestras pertenencias.

- *PROCESS bolsa_meter(obj)*: Añade el objeto "obj" a la bolsa. Para añadir un objeto, recorremos la bolsa en busca del *ID*, ya que en *DIV* no puede haber dos procesos simultáneos con el mismo *ID*, y en nuestra bolsa tampoco debemos meter algo que ya está dentro. Si se encuentra este *ID* (ya está en la bolsa), no hacemos absolutamente nada, en caso contrario, lo añadimos a la bolsa, pasando a ser el último y nos esforzamos para visualizarlo, calculando un objeto que al ser actual nos permita ver el último.
- *PROCESS bolsa_sacar(obj)*: Extrae el objeto "obj" de la bolsa.
- *PROCESS bolsa_dibuja()*: Dibuja la bolsa adecuadamente. En nuestro juego sólo pueden visualizar-

Nuestra bolsa es capaz de contener hasta 35 objetos. pero sólo podrán visualizarse 14 a la vez. en dos filas de siete objetos



Los objetos pueden estar dispersos por todo el escenario.



Tenemos pocos utensilios todavía.



Cada objeto tiene un uso.



Nos vamos de compras.

se 14 objetos al mismo tiempo (en 2 filas de 7 objetos). Para conseguir este efecto se visualizan los objetos desde *Actual* hasta *Actual* + *14* (si existen), quedando el resto dibujados fuera de los límites de la pantalla.

- *PROCESS bolsa_arriba(file)*: Este proceso añade una flecha al menú de acciones y desplaza el contenido de la bolsa al detectar el *click* del ratón sobre él.
- *PROCESS bolsa_abajo(file)*: Similar a *bolsa_arriba*.

Es necesario conocer el programa MOD si estamos iniciandonos en la programación

Un objeto, al ejecutarse la acción "coger", cambia su gráfico y se autoañade a la bolsa. Si por el contrario recibe la orden adecuada, vuelve a cambiar su gráfico y se autoextrae de la bolsa.

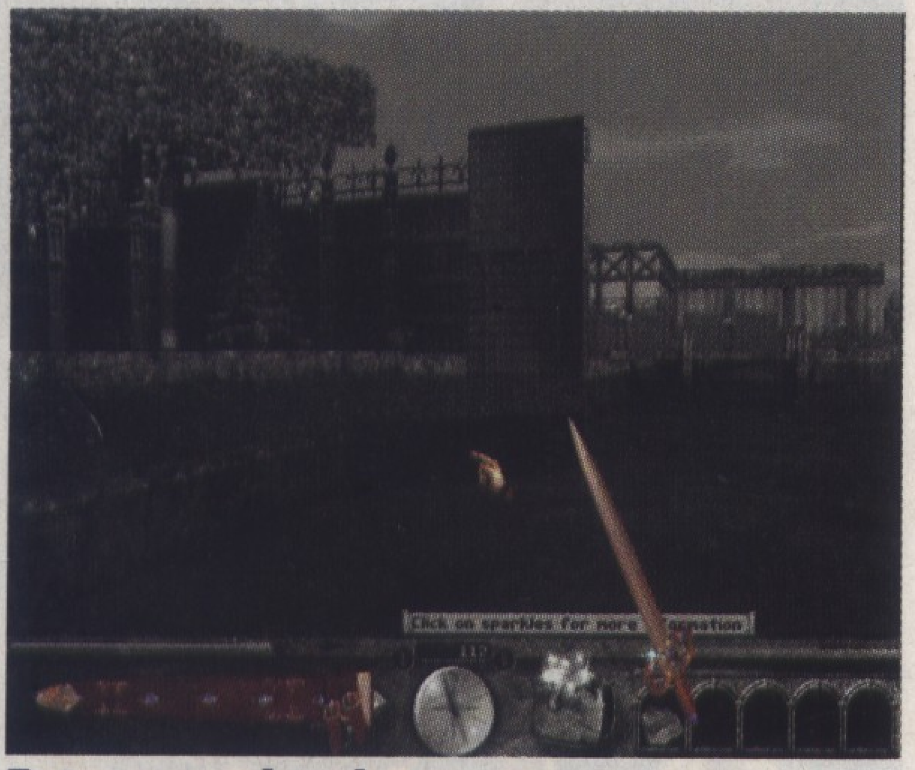
A mod B

Para los que se inician en la programación les será necesario conocer el funcionamiento de *MOD*. Este operador matemático nos simplifica a menudo algunas tareas. *MOD* nos devuelve el resto de una división entera. Al dividir dos números el resto ha de ser siempre inferior al divisor. Según esto, siempre que trabajamos con un rango de valores podemos conseguir que éste sea un rango de *0* a *n-1*, teniendo un total de *n* valores posibles. Al hacer *a MOD b*, el resultado siempre será menor que *b*. Si hacemos *x MOD 4*, el resultado, dependiendo de *x*, será 0, 1, 2 ó 3. Esto nos permite determinar múltiplos enteros perfectos y hacer cálculos para seguimientos circulares:

Circulo ascendente:
i = (i+1) mod n (n=4, i = 0, 1, 2, 3, 0, 1 ...)

Circulo descendente:
i = (n+i-1) mod n (n=4, i=3, 2, 1, 0, 3, ...)

Múltiplo perfecto:
0 == i mod n (n=4, i=148 : TRUE)



Esta espada ahora es nuestra.

Procedimiento que hace de comodín para las acciones generales.

```
PROCESS acciones(accion_id);
 PRIVATE
  n;
  ac;
 BEGIN
  // Si se especific¢ una
acci¢n, se hace seg£n ella, si no,
se toma
  // la acci¢n seleccionada
  IF (accion_id == o_none)
   ac = Accion.id_accion;
  ELSE
   ac = accion_id;
  END;

  // Generamos un n§ aleato-
rio que debe ser redondeado
   n = rand(0, 30);
   SWITCH (ac)
    CASE o_none:  END;
    CASE o_Ir_a:  END;
    CASE o_Hablar:
     SWITCH (n mod 2)
      CASE 0: habla("Ser¡a
perder el tiempo"); END;
      DEFAULT: habla("¨T£
crees que deber¡a?"); END;
     END;
    END;
    CASE o_Mover:
     SWITCH (n mod 3)
      CASE 0: habla("No
alcanzo"); END;
      CASE 1: habla("Parece
muy pesado"); END;
      DEFAULT: habla("No
creo que deba hacer eso"); END;
     END;
    END;
    CASE o_Mirar:
     habla("No veo nada en
especial");
    END;
    CASE o_Coger:
     SWITCH (n mod 4)
      CASE 0: habla("No lo
necesito"); END;
      CASE 1:
habla("Podr¡an echarlo en falta");
END;
      CASE 2: habla("Es
muy pesado"); END;
      CASE 3: habla("Mejor
dejarlo ah¡"); END;
      CASE 4: habla("No lo
necesito"); END;
      DEFAULT: habla("No
puedo llevarlo"); END;
     END;
    END;
    CASE o_Dar:
     habla("Mejor no");
    END;
    CASE o_Abrir:
     habla("Parece no abrir-
se");
    END;
    CASE o_Cerrar:
     habla("Parece no
cerrarse");
    END;
    CASE o_Usar:
     SWITCH (n mod 4)
      CASE 0: habla("No se
que hacer con eso"); END;
      CASE 1: habla("Parece
no funcionar"); END;
      CASE 2: habla("Hacer
eso no tiene sentido"); END;
      DEFAULT:
habla("Mejor no"); END;
     END;
    END;
    DEFAULT:
     habla("No entiendo lo
que quieres hacer");
    END;
   END;
   des_objeto();
  END;
```



Mejor darle el papel a ella.



Otro escenario para explorar.

La práctica

En el número anterior nuestros objetos quedaron aún muy sosos. Ya podemos interactuar con ellos, pero, ¿y si queremos combinar objetos de la forma *usar objeto1 con objeto2*? Llamaremos *primario* a objeto1 y *secundario* a objeto2. Debemos distinguir ahora dos tipos de objetos: los combinables y los no combinables.

1. Objetos no combinables: la manera de actuar de un objeto no combinable con otro será:

Si no hay objeto secundario: ejecutamos.

Si hay objeto secundario: si somos secundario pasamos el control al primario. Si somos primario, damos error.

2. Objetos combinables con otros objetos: cuando un objeto puede combinarse con otro, lo que debemos hacer es lo siguiente:

Si no hay objeto secundario: nos preparamos para esperar al secundario. No seguimos ejecutando. Si hay objeto secundario: Podemos hacer tres cosas:

- Determinar quién es el otro objeto (si somos primario o secundario).
- Sabemos que hacer con el otro objeto. Hacemos lo que tengamos que hacer.
- No sabemos que hacer con el otro objeto. Si somos secundario, pasamos el control al primario. Si no somos secundario, damos error.

Es muy importante al combinar objetos asegurarse de que ambos no estén en la bolsa de objetos. Si lo estuviesen, se deben sacar. El objeto nuevo resultante puede añadirse a la bolsa si se quiere. Cuando se combinan dos objetos (A y B), puede ocurrir lo siguiente:

1. A se modifica y B muere. Debe ejecutarse siempre A para que mate a B. Ejemplo: si usamos una chincheta en un tablero de corcho, el objeto tablero se modifica, mientras que la chincheta desaparece.

2. A se modifica y B permanece. Preferiblemente debe ejecutarse A. Ejemplo: usamos una botella de agua con un vaso vacío. El objeto vaso se modifica, pero la botella no desaparece.

3. A y B se modifican. Ejemplo: en el caso anterior, el objeto botella se modifica porque ya no está completamente llena.

Nuestro programa

Para realizar todo esto hemos tenido que modificar el gráfico del menú, eliminando los cuadros de la bolsa de objetos, ya que ahora sí saldrán objetos. Las flechas para subir y bajar son gráficos independientes para los procesos *bolsa_arriba()* y *bolsa_abajo()*.

> Los objetos pueden combinarse entre sí y dar lugar a otros objetos

En esta ocasión, para probar nuestro código hemos incluido siete arcos, siete flechas, siete dianas y siete trípodes. El comportamiento de los objetos es:

Flecha: sólo se puede usar con arco y cogerse.

Arco con flecha: sólo puede usarse con la diana sobre trípode, volviendo a ser arco.

Diana: Sólo se puede coger y usar con el trípode para darle consistencia, entonces se convierte en diana sobre trípode, desapareciendo de nuestra bolsa de objetos si la teníamos.

Trípode: Sólo se puede usar con la diana.

Miguel A. Barroso



Los objetos pueden conseguirse en los lugares más insospechados.

Nota sobre los ficheros fuente.

Los ficheros fuente del CD están preparados para ser instalados en el directorio \div\cag\04 en la unidad donde tengas instalado *DIV Games Studio*.

DIVnet



DIVertida y DIVersa

Nace la primera revista electrónica dedicada al mundo de la programación con DIV y DIV2. Su nombre DIVnet. Una estupenda página web, totalmente independiente, y que se convertirá, seguro, en referencia obligada para todos aquellos interesados en el programa DIV. Otra web para añadir a vuestra libreta de direcciones.

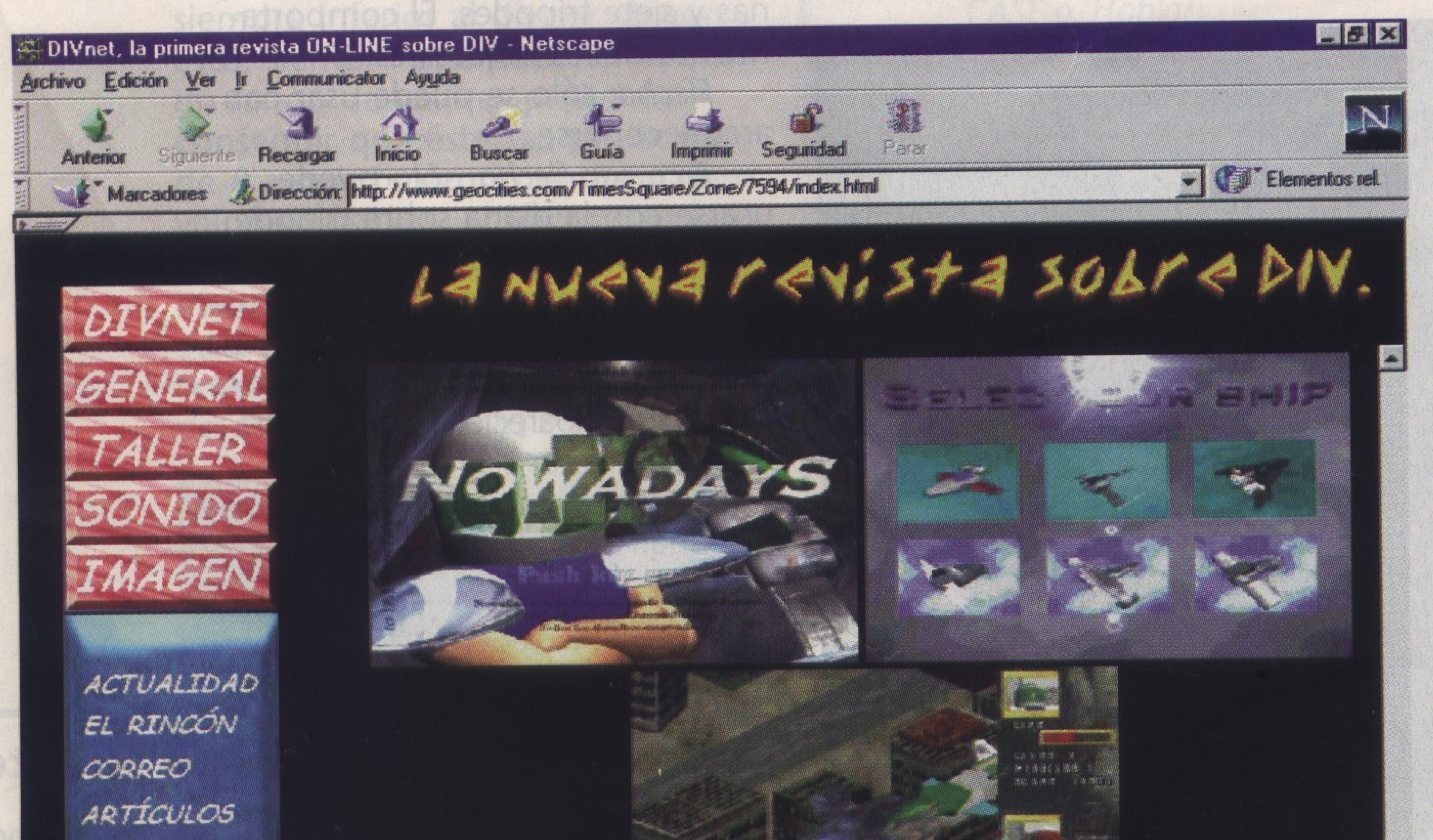
Era sólo cuestión de tiempo el que apareciera en el proceloso mar de Internet una página en forma de revista electrónica dedicada al mundo DIV. Y su nacimiento no se ha producido de cualquier manera. Os podemos asegurar que esta revista es completísima, y eso que sólo están en su número cero. Nuestra felicitación a los creadores de esta magnífica idea. Desde estas páginas os aconsejamos que le echéis una ojeada en cuanto podáis. Eso si disponéis de la posibilidad de conectaros a Internet, si no, no os preocupéis, tan buena nos ha parecido que incluimos completo el número cero en el CD-Rom que acompaña a la revista. Ahora vamos a destriparla un poco, aunque seguro que alguno ya habrá tecleado la dirección *web* en su navegador favorito.

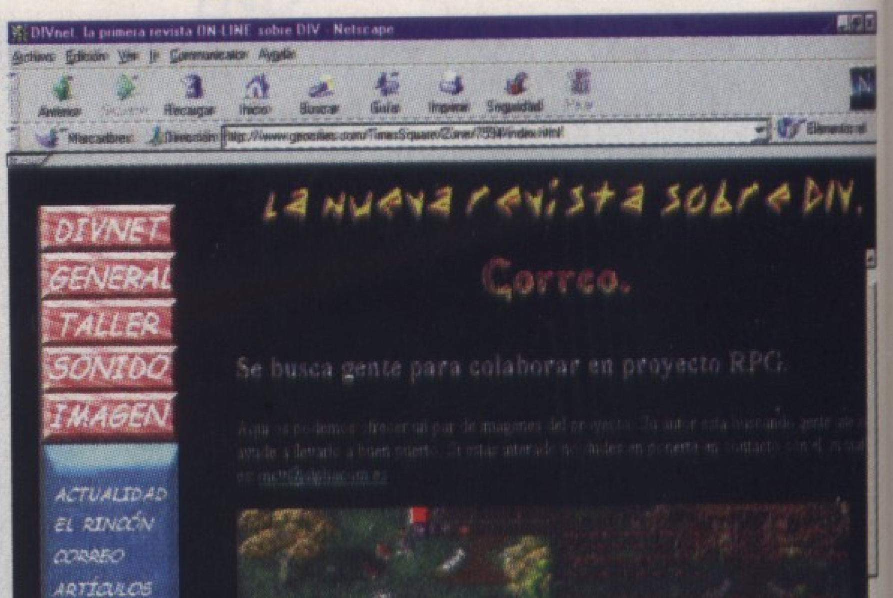
DIVnet por dentro

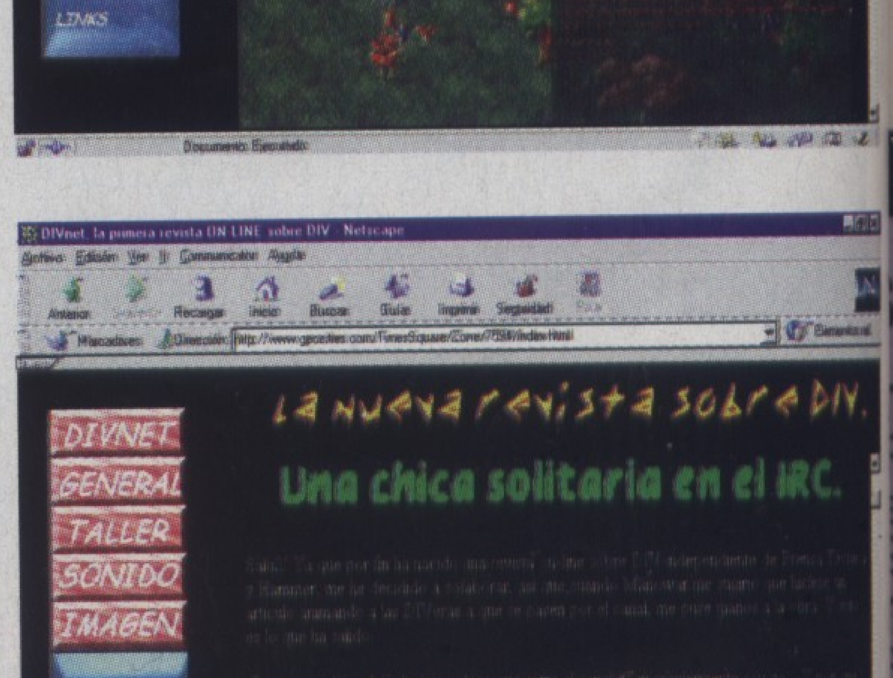
Esta revista electrónica está dedicada por completo al "fantabuloso" mundo de DIV. Como dicen los mismos creadores de la publicación, este universo en torno al desarrollo de juegos con la herramienta DIV esta creciendo día a día. Suponemos que esto es debido a que cada vez hay más personas que no se conforman sólo con jugar y jugar a sus títulos favoritos. Las inquietudes de todos aquellos que quieren dedicarse profesionalmente a la programación de juegos están siendo colmadas, en gran parte, con la aparición de DIV y DIV 2. Esto se tenía que reflejar en la Red tarde o temprano.

DIVnet, como toda publicación que se precie, tiene una editorial dedicada, como no podía ser menos en este número cero, a la presentación de la revista. El objetivo de DIVnet es, aparte de convertirse en un sitio obligado de visita para todos los interesados en saber las últimas novedades en torno a DIV, ayudar a los que empiezan a interesarse en aprender a desarrollar sus propias obras. La programación de videojuegos, aunque es una actividad poco desarrollada en España, salvo honrosas excepciones, es una de las actividades profesionales que más futuro tienen y en algunos países el sector es uno de los de más alto crecimiento económico.

La revista tiene una sección, llamada "actualidad" dedicada a las







CATÁLOGO DE LIBROS DE PRENSA TÉCNICA

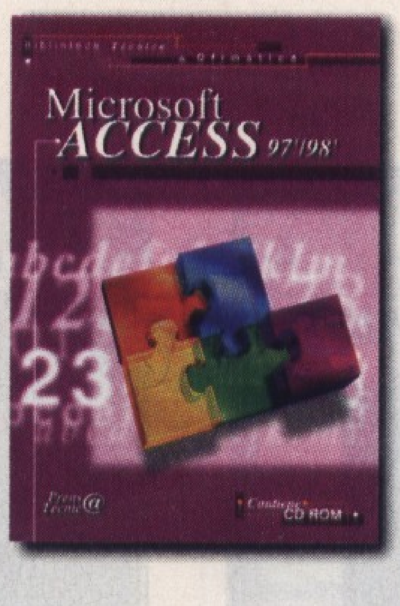
Cómo pasar 10 minutos divertidos en el Chat



La comunicación en tiempo real hablada y escrita, la videoconferencia y el IRC dejarán de tener secretos para ti con esta obra dedicada a hacer más divertido Internet cuando chateamos.

2.995 ptas. Incluye CD-ROM.

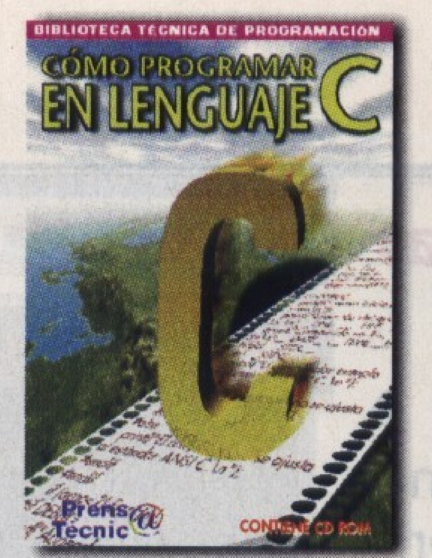
Microsoft Access 97-98



Microsoft Access, una de las aplicaciones más populares hoy en día, que ha logrado consolidarse como un auténtico estándar entre los gestores de bases de datos para entornos gráficos como es el caso de Windows.

2.995 ptas. Incluye CD-ROM.

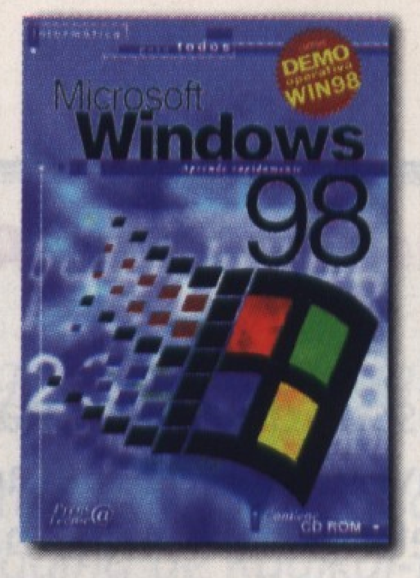
Cómo Programar en Lenguaje C



El lenguaje C es el modelo de programación por excelencia, el más utilizado para todo tipo de aplicaciones. Este libro le introducirá en la programación C de una manera clara y sencilla.

2.995 ptas. Incluye CD-ROM.

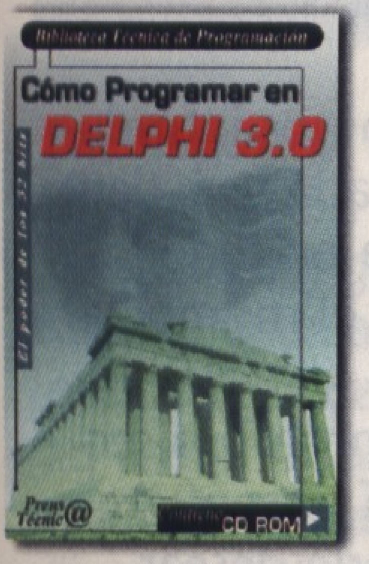
Cómo trabajar con Windows 98



Descubra todas los avances del nuevo sistema operativo de Microsoft y domine en poco tiempo todas las formas de trabajar con él. Incluye un CD-Rom con una demo completamente operativa del programa.

2.995 ptas. Incluye CD-ROM.

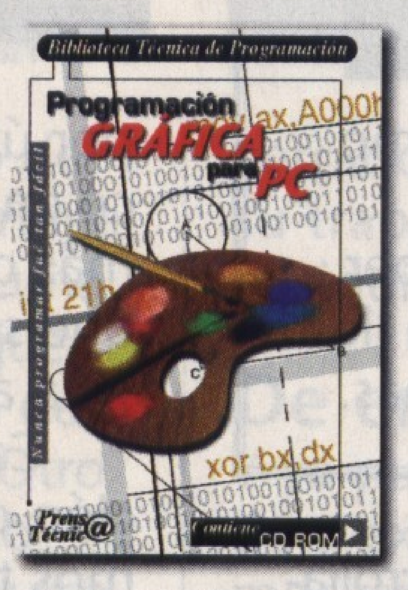
Cómo Programar en Delphi.



Delphi se puede considerar como uno de los entornos de programación visual más poderosos y a la vez más fáciles de utilizar y aprender. Esta obra está orientada a nivel principiante e intermedio.

2.995 ptas. Incluye CD-ROM.

Programación Gráfica para PC



Es una obra dedicada a todos aquellos que quieran adentrarse en el apasionante mundo de la programación visual. Se puede aprender, de una forma rápida y sencilla, a explotar todas las capacidades gráficas de su ordenador.

2.995 ptas. Incluye CD-ROM.

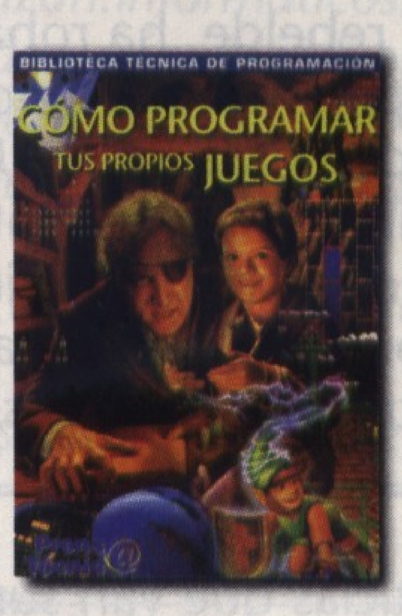
Cómo Programar en Visual Basic



Con Visual Basic 5.0 creará aplicaciones Windows rápidamente, trabajando con una de las herramientas de desarrollo más potentes del mercado. Con este libro conocerá sus propiedades y conceptos básicos de programación.

2.995 ptas. Incluye CD-ROM.

Cómo Programar tus Propios Juegos



Introduce al lector de forma sencilla en el mundo de la programación de videojuegos, utilizando las técnicas más avanzadas en este campo recorrido por la historia de los videojuegos.

2.995 ptas. Incluye CD-ROM.

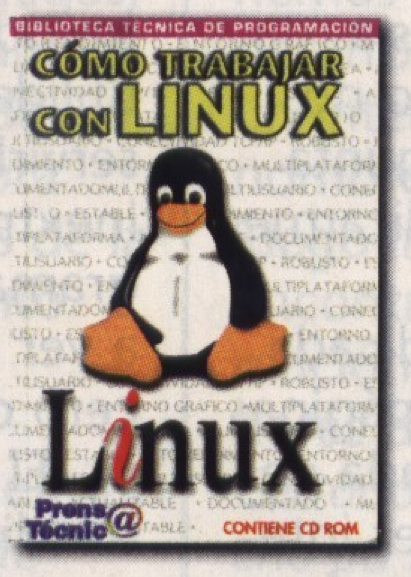
Cómo trabajar con Corel Draw 8



Herramienta de trabajo que puede ser utilizada tanto a nivel profesional como particular. Manual práctico con el que acceder a las múltiples posibilidades de Corel-Draw.

2.995 ptas. Incluye CD-ROM.

Cómo trabajar con Linux



Explica cómo comenzar a usar Linux y será de utilidad para sacar más partido a su instalación. El sistema ofrece multitarea, potente entorno gráfico, alto rendimiento y conectividad.

2.995 ptas. Incluye CD-ROM.

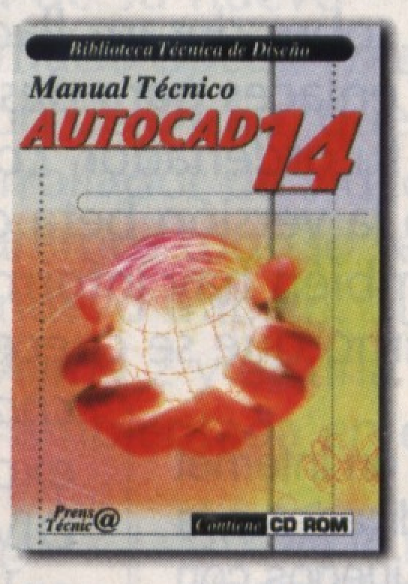
Directorio Fundamental de Internet



Directorio Fundamental de Internet es una completa guía, en la que podrá encontrar las direcciones más útiles al llevar a cabo la compleja tarea de navegar buscando un tema concreto.

2.995 ptas. Incluye CD-ROM.

Manual Técnico AutoCAD 14



Descubre la última versión de AutoCAD 14, un programa de dibujo de propósito general, mayoritamente difundido en el ámbito del diseño asistido por ordenador.

2.995 ptas. Incluye CD-ROM.

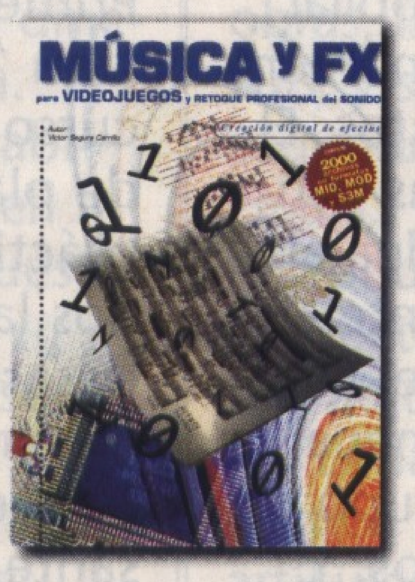
Manual Técnico Lightwave



Introduce al lector en el mundo del modelado con Lightwave de una forma sencilla y práctica: creación de efectos atmosféricos, animaciones, etc. Incluye ejemplos prácticos.

2.995 ptas. Incluye CD-ROM.

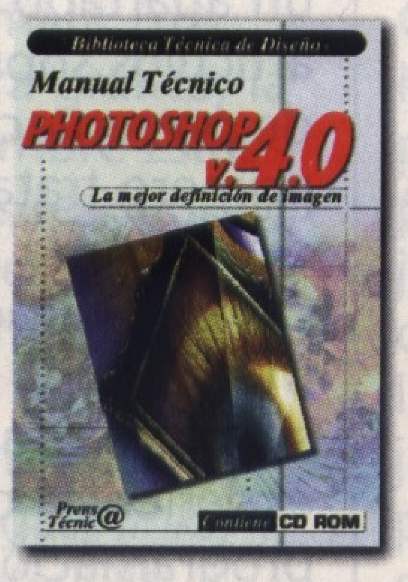
Música y FX para videojuegos



Medios Técnicos necesarios para la elaboración de la banda sonora de un videojuego, así como los efectos de sonido. Todo ello acompañado de una serie de ejercicios prácticos que el lector podrá ir realizando a medida que avanza el libro.

2.995 ptas. Incluye CD-ROM.

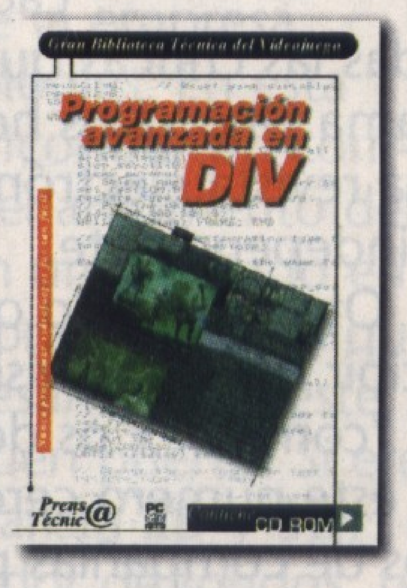
Manual Técnico PhotoShop 4.0



Photoshop es el software de retoque fotográfico por excelencia y el programa más utilizado por los profesionales del diseño. Esta obra permitirá al lector adentrarse en el mundo de la imagen digitalizada, su tratamiento, etc.

2.995 ptas. Incluye CD-ROM.

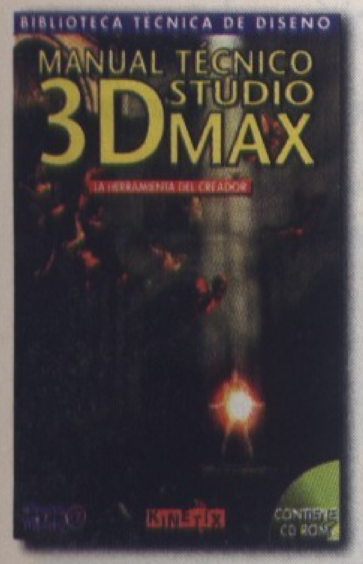
Programación avanzada en DIV



Conozca los últimos avances en programación bajo DIV Games Studio, con ejemplos, trucos, modos de trabajo y una galería de direcciones web y canales de IRC sobre el mundo DIV.

2.995 ptas. Incluye CD-ROM.

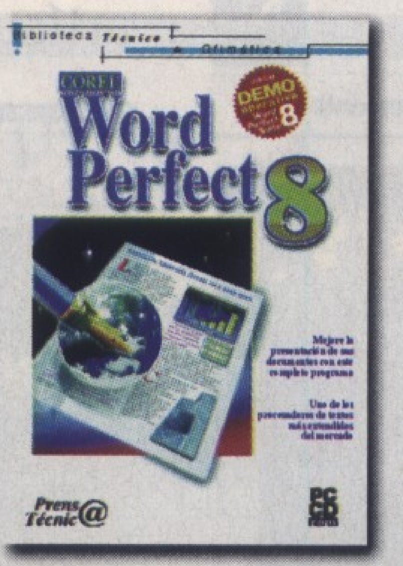
Manual Técnico 3D Studio Max



La obra definitiva tanto para el usuario novel como para los ya iniciados, que lograrán dominar esta herramienta en poco tiempo. Incluye CD-Rom repleto de utilidades, modelos y texturas para trabajar con 3D Max.

2.995 ptas. Incluye CD-ROM.

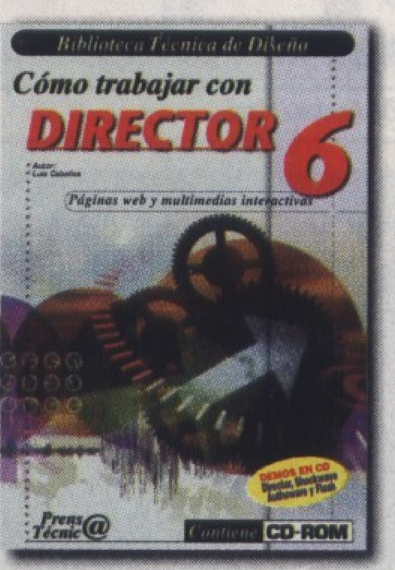
Manual Práctico de Corel WordPerfect 8



El dominio de uno de los programas de tratamiento de texto está a tu alcance con este libro que repasa todas las funciones. Nunca resultó tan sencillo sacarle el máximo partido a WordPerfect.

2.995 ptas. Incluye CD-ROM.

Cómo trabajar con Director 6



Páginas web y creación multimedia con este libro, que recorre uno a uno los aspectos fundamentales de Director 6, un programa que permite trabajar con sonidos, animaciones, texto, vídeo digital, etc.

2.995 ptas. Incluye CD-ROM.

Puede adquirir estos productos llamando al nº de teléfono 91 304 06 22 o en los distribuidores oficiales indicados abajo.

ISTRIBUIDORES

ASTILLA Y LEÓN:
ORES GARCÍA LIBROS, S.L. C/ Pintores, 5. ares de la Reina, 37184. Salamanca.
f: 923 23 02 06, fax: 923 25 31 17.

ALENCIA-CASTELLÓN:
USIO GENERAL DE LLIBRERIA, S.L. Avd. Ausias rch, 34 Bajo. Tavernes Blanques. 46016 Valencia.
f: 961 85 18 28, fax: 961 85 88 91.

ATALUNYA Y BALEARES:
STRIBUCIONES PRÓLOGO, S.A. C/ Mascaro, 5. 08032 Barcelona.
f: 934 56 20 00, fax: 934 56 95 06.

ADRID Y CASTILLA LA MANCHA:
STRIFORMA, S.A. C/ Abtao, 25 Nave B. 007, Madrid.
f: 915 01 47 49, fax: 915 01 48 99.

ALICANTE-MURCIA-ALBACETE
GAIA LIBROS, S.L. C/ Sagitario, 7-9. 03006, Alicante
Telf: 965 11 05 16, fax: 965 11 41 26.

GALICIA
GRIAL LIBROS, S.L. Pol. Ind. Tambre, Via Diesel, 4. 15890 Santiago de Compostela.
Telf: 981 58 97 54, fax: 981 57 59 04.

ARAGÓN-LA RIOJA-SORIA
ICARO DISTRIBUIDORA, S.L. Pol. El Plano, C/ A,39, Mª de Huerva. 50430 Zaragoza.
Telf: 976 12 63 33, fax: 976 12 64 95.

ANDALUCÍA ORIENTAL
NADALES LIBROS, S.A. Camino Bajo S/N, Armilla. 18100 Granada.
Telf: 958 55 08 03, fax: 958 57 15 56.

PAIS VASCO-NAVARRA
TOMÁS ELLACURIA, S.A. Pol. Legizamon, C/ Nervion, 1 bajo. 48450 Etxebarri, Vizcaya.
Telf: 944 26 12 50, fax: 944 26 12 88.

ASTURIAS-SANTANDER
NORTE PROMOCIONES, S.A. C/ Pérez Ayala, 10. 32208 Gijón.
Telf: 985 14 70 30, fax: 985 38 88 14.

EXTREMADURA
UNIDISA, S.A. Pol. Ind. El Nevero, Complejo La Mar, Nave 10. 06006 Badajoz.
Telf: 924 27 51 92, fax: 924 27 56 01.

ISLAS CANARIAS
TROQUEL DISTRIBUCIONES. Urb. Montaña Blanca, Parcela 31A, Arucas. 35415 Gran Canaria.
Telf: 928 62 17 80, fax: 928 62 17 81.

ANDALUCÍA OCCIDENTAL
DYD/UNIDISA, S.A. Pol. El Pino, Parcela 8E Nave 21-22. 41016 Sevilla.
Telf: 954 51 63 33, fax: 954 25 50 98.

DE VENTA EN LIBRERÍAS

Proyecto Estar Guars.



Todos están de mala uva, es la guerra civil. Las naves rebeldes, atacando desde una base secreta (Rabin 4.5), han ganado la primera batalla contra el malvado, ruin y pendenciero Imperio Colacao.

En el transcurso de la batalla, "Carne de Cañón", un rebelde, ha robado los planos de la última superarma *que-te-cagas de chunga*... la "Baticao de la Muerte". Un terror *tessnológico* con el suficiente poder como pá que la diñen planetas enteros.

Perseguida por agentes imperiales, la princesa Leia vuelve a su keli sin saber que algún pringao le ha colao los planos robados... unos planos que pueden llevar a la destrucción total si cayeran en manos de las multinacionales niponas.

noticias que van surgiendo dentro del mundo de la programación y desarrollo de juegos. En este primer número podemos encontrar artículos sobre desarrolladores, compañías, enlaces para descargar ficheros en tu ordenador, como por ejemplo la versión beta del juego *Enemy*. También hay información sobre lo último que se cuece en el canal de *chat* dedicado exclusivamente a DIV. Incluso hay un enlace a una página llamada "Radikal-Div" dedicada a recoger todas las críticas que suscita el programa DIV. Ya os hemos dicho que está revista electrónica es totalmente independiente.

Las páginas en Internet dedicadas a la programación de juegos con la herramienta DIV y DIV 2 son ya multitud, aquí hay un ejemplo de las mismas

Otro apartado que seguro va a recibir bastantes visitas es la sección de "correo". Y os decimos por qué. En este número cero ya hay anuncios de compañías pidiendo desesperadamente grafistas y desarrolladores para un juego de rol que está en su fase de proyecto. Ya os hemos dicho que la programación de videojuegos es una de las opciones a considerar seriamente cuando nos planteamos la inevitable pregunta de ¿a qué narices puedo dedicarme para ganar algo de dinero?

La ineludible sección dedicada a recopilar enlaces de otras páginas *web* también esta representada en esta revista. Y con direcciones interesantes que luego veremos más detenidamente.

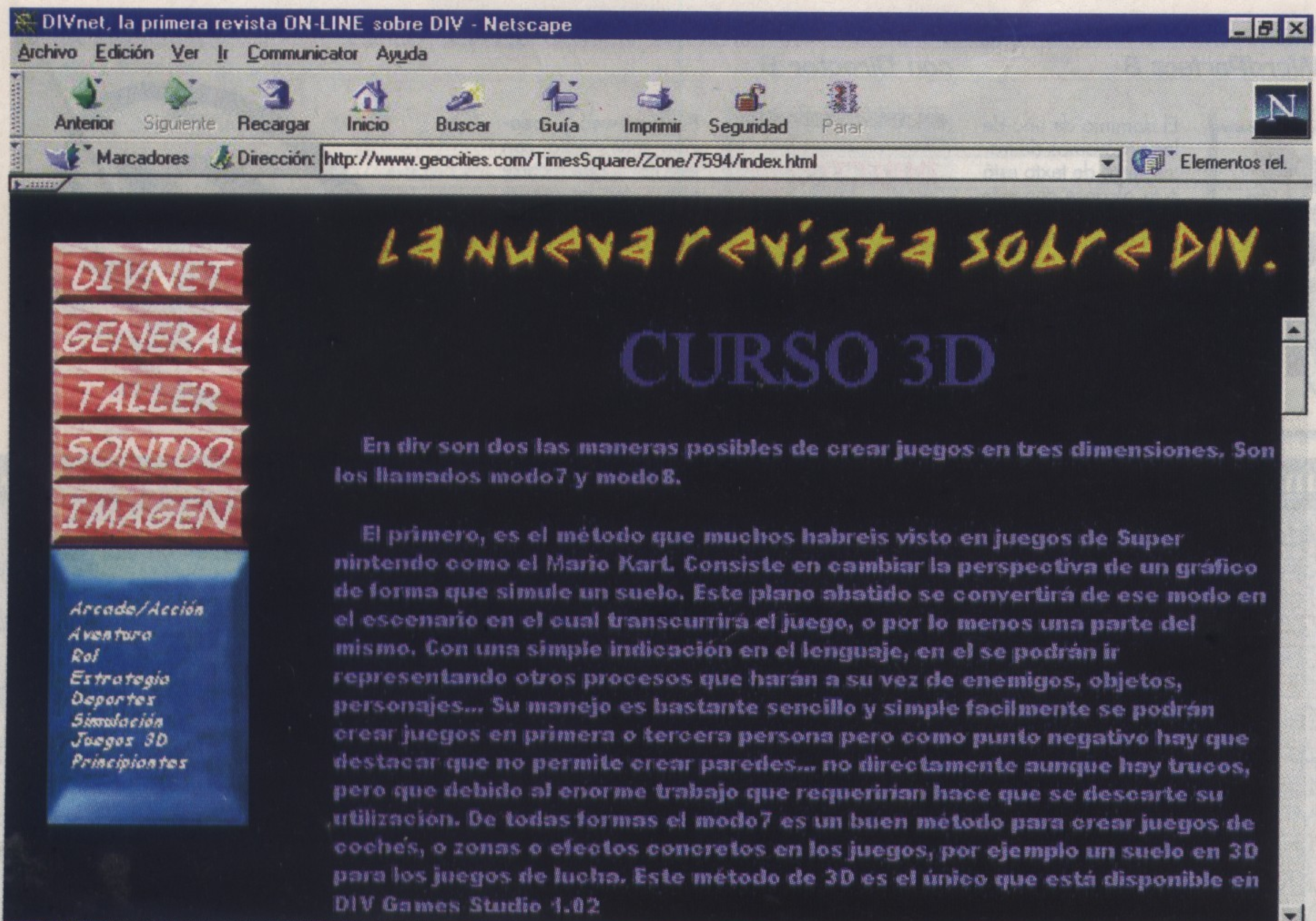
También podemos encontrar un apartado dedicado a recopilar artículos sobre DIV. En este número de presentación hay tres textos. El primero trata de las vivencias cotidianas de la única chica asidua al canal dedicado exclusivamente al mundo DIV. Muy interesante. El segundo trata, de forma amena, la experiencia de formar un grupo de programación de videojuegos. Y,

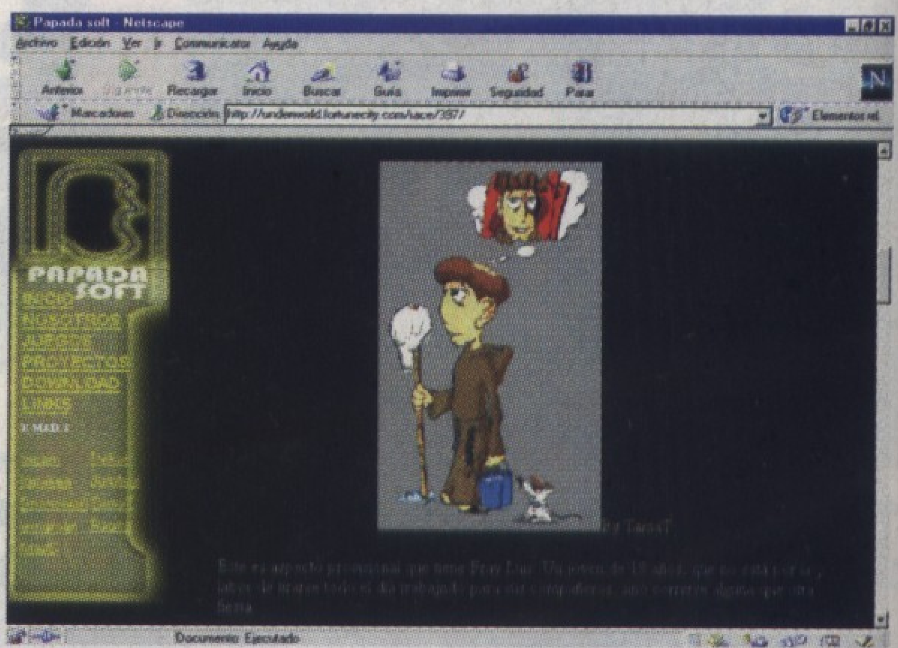


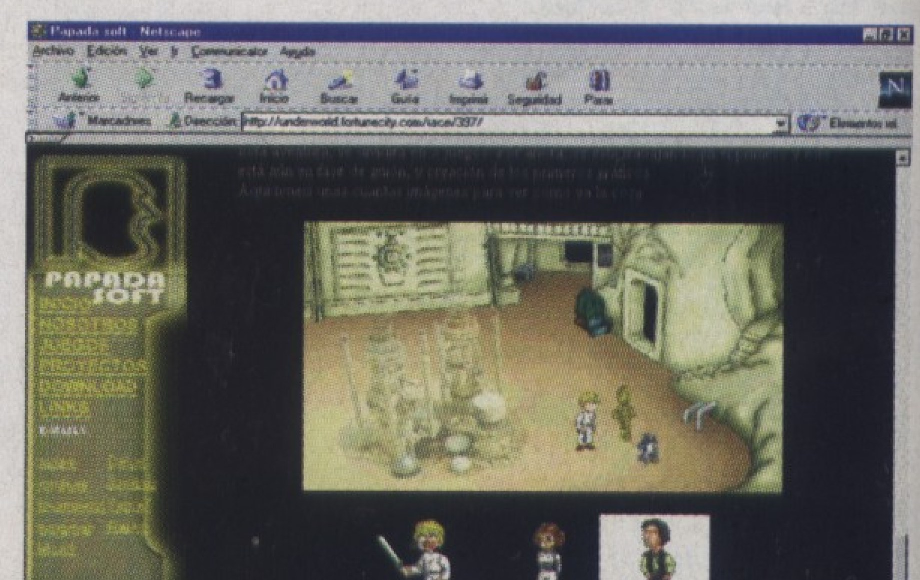
por último, el tercero nos introduce en un tema que cada vez tiene más gente interesada en todo lo que se publica sobre el mismo: la vida artificial.

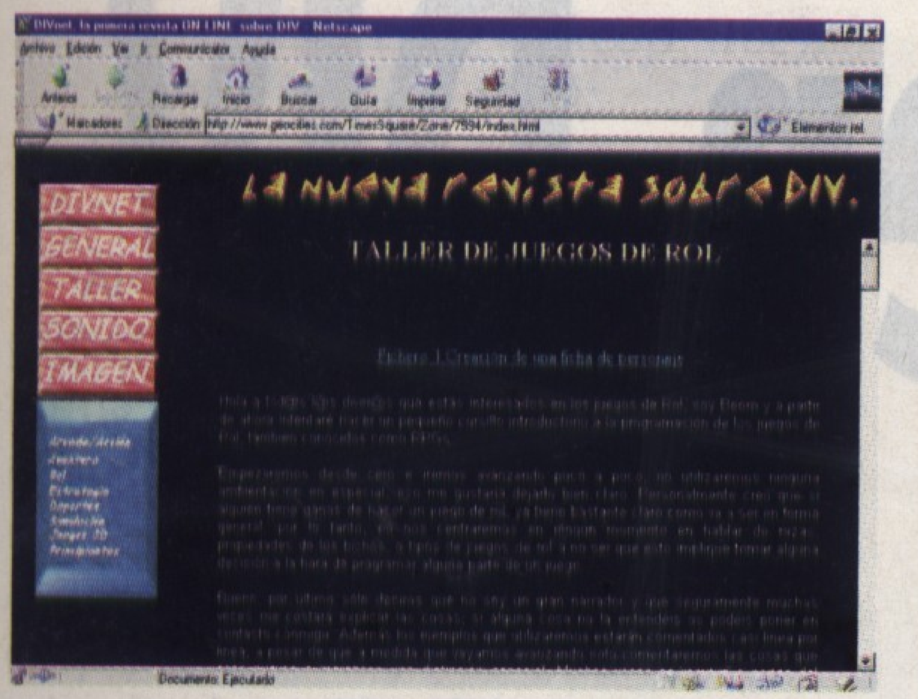
Capítulo aparte merece la sección de "taller". Aquí encontraremos todo lo que puede desear cualquier aficionado al programa DIV. Sobre todo los que empiezan a introducirse en este apasionante mundo descubrirán todo tipo de utilidades y ayudas que pueden servirles para no perderse en este inextricable bosque que es la programación de videojuegos. Vamos a ver que nos ofrece el número cero de DIVnet en esta sección.

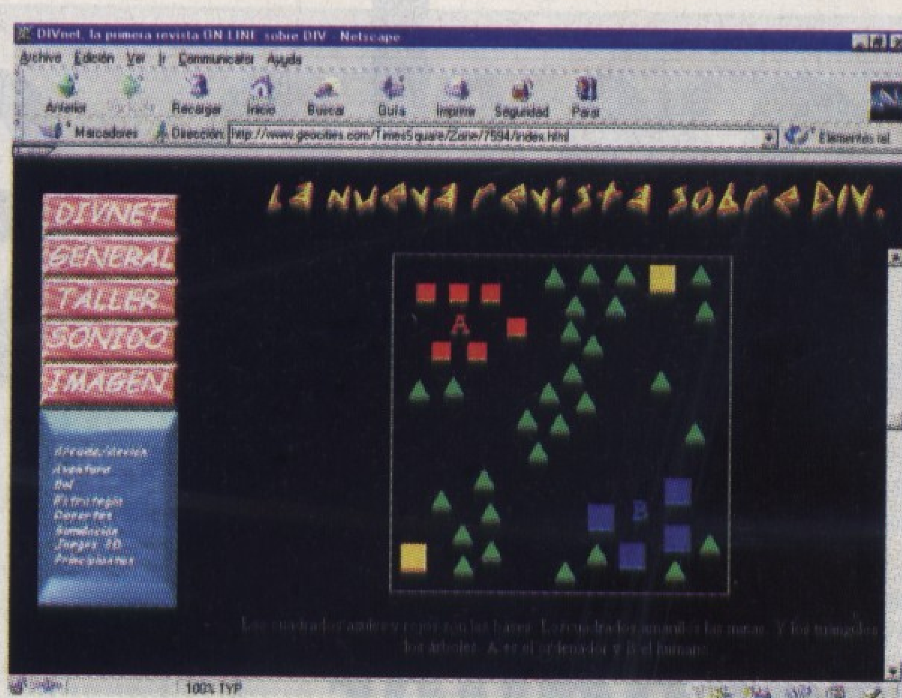
El apartado de "taller" esta dividido en varias partes, cada una dedicada a los géneros habituales en los que podemos clasificar cualquier título lúdico. Primero nos topamos con la sección "arcade/acción" con un curso de programación de juegos de plataformas, con ejemplos prácticos incluidos. En "aventura" nos las veremos con un cursillo de desarrollo de juegos tipo aventuras gráficas. En "rol" hay un taller de juegos de rol bastante interesante. También hay un hueco para los aprendices de generales sin mando en plaza en el apartado "estrategia", con un completo artículo de introducción al tema.











En "deportes" hay un ejemplo de programación de un juego perteneciente a este grupo, y además una sección dedicada a revisar los juegos clásicos dentro del género de las competiciones deportivas. Otro rincón interesante es el llamado "juegos 3D" con un práctico curso de programación 3D. Y por último, para auténticos novatos en el tema de la programación de videojuegos, una sección llamada "principiantes" que contiene un curso de fundamentos de programación. Habrá también algo para los amantes de la simulación pero por el momento tendrán que esperar hasta la siguiente edición de la revista.

Otra posibilidad que ofrece esta página *web* es la posibilidad de realizar un curso de diseño 2D y un curso de grafismo 3D vía módem. Todo está en la sección de imagen. Ya no hay escusas si no tienes dinero para pagarte una academia. Internet es todo un pozo de sabiduría del que podemos sacar el agua necesaria para apagar nuestra sed de conocimiento. Incluso podemos ducharnos con él. Esta dirección es buena prueba de ello.

Reiteramos nuestra enhorabuena a los creadores de esta estupenda *web* y les lanzamos, desde estas páginas, un aplauso cerrado para que perseveren en su trabajo. Estamos deseando leer el siguiente número.

De enlace a enlace

Y ya puestos no nos hemos resistido a explorar los *links* que podemos encontrar en la sección homónima de la revista DIVnet.

La primera página donde hemos caído es en la del grupo de desarrollo "PapadaSoft" que tienen un interesante logo, acorde con el nombre de la empresa. Desde luego inusual sí que es. Podemos encontrar en esta dirección información y noticias sobre los creadores de los juegos llamados *Caso Cerrado I, Caso Cerrado II* y *Me marcho pa Francia 98.* Todos los juegos pueden bajarse para que podamos evaluar el trabajo de estos castizos creadores. En proyecto tienen dos obras, la primera se llamará *Fray Luis*, y será una aventura gráfica; la segunda llevará por nombre *Estar Guars,* y no nos hemos resistido a reproducir el guión de este futuro proyecto. Lo podéis ver en el cuadro adjunto del mismo título.

En la página de Benko encontramos noticias, descargas, *links*, y artículos en torno a DIV y DIV 2.

Cristal Wave es un grupo desarrollador de juegos con el entorno DIV. Desde su dirección en Internet se puede bajar la última versión Beta de su nuevo juego de rol llamado *Speak Freely.*

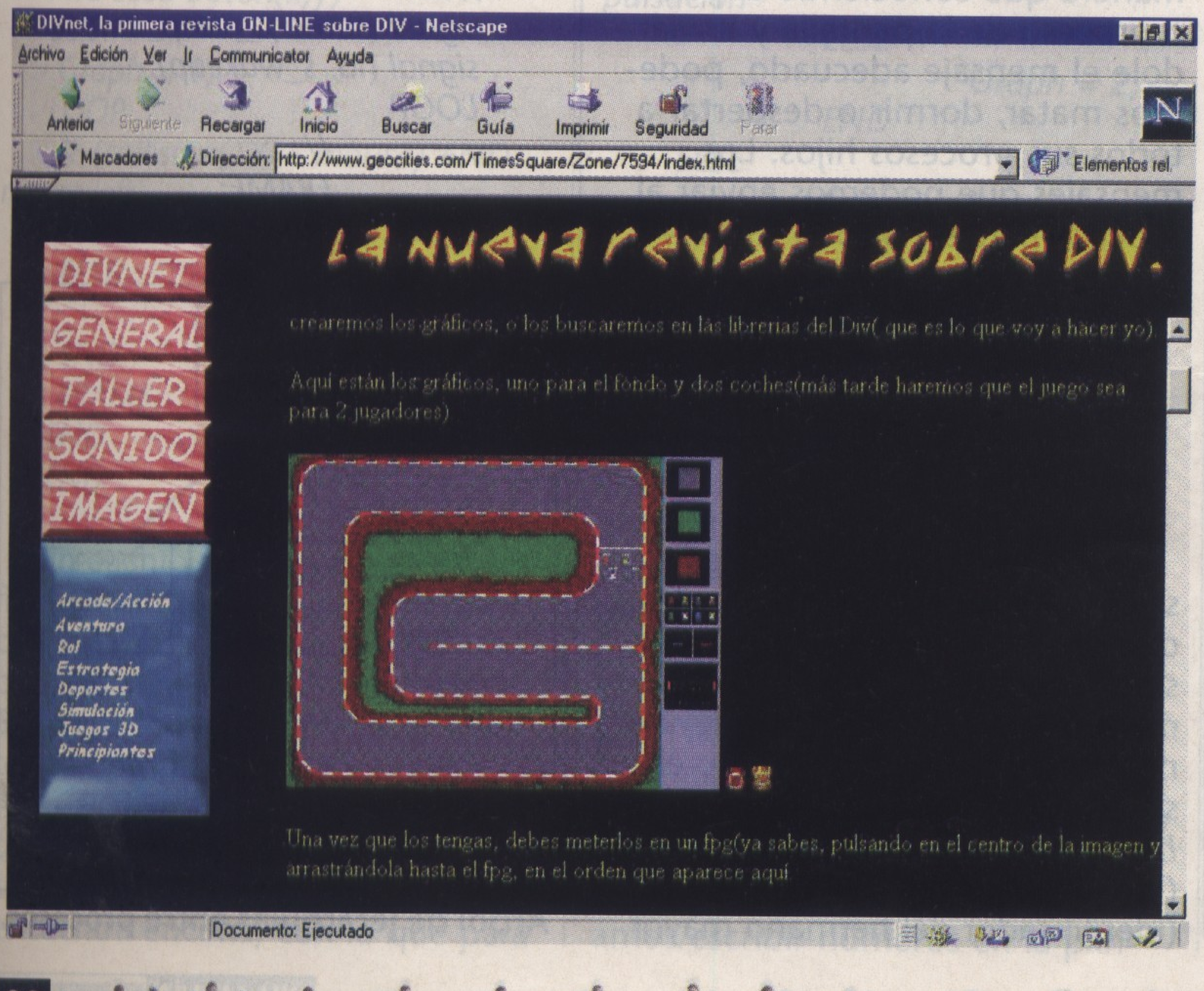
En la dirección de Dizan, que es otro grupo desarrollador de videojuegos, encontraremos todo lo relacionado con su nuevo trabajo. Su obra se llamará *Alpha 2* y es una aventura gráfica ambientada en el espacio exterior en el año 2050. Este juego está siendo desarrollado con el programa DIV Games Studio de la empresa Hammer Technologies. En esta página podéis encontrar un completo resumen del argumento del juego. Y también un apartado en el que piden colaboradores para que la empresa de desarrollo de este juego llegue a buen puerto. Si eres programador, grafista 2D y 3D, dibujante, ilustrador de fondos, técnico de efectos de sonido, escritor o guionista, tienes una oportunidad de colaborar con ellos en el desarrollo de *Alpha 2*. Incluso piden dobladores para poner voz a los personajes. Date una vuelta por su página y, quién sabe, puede que seas su próxima adquisición.

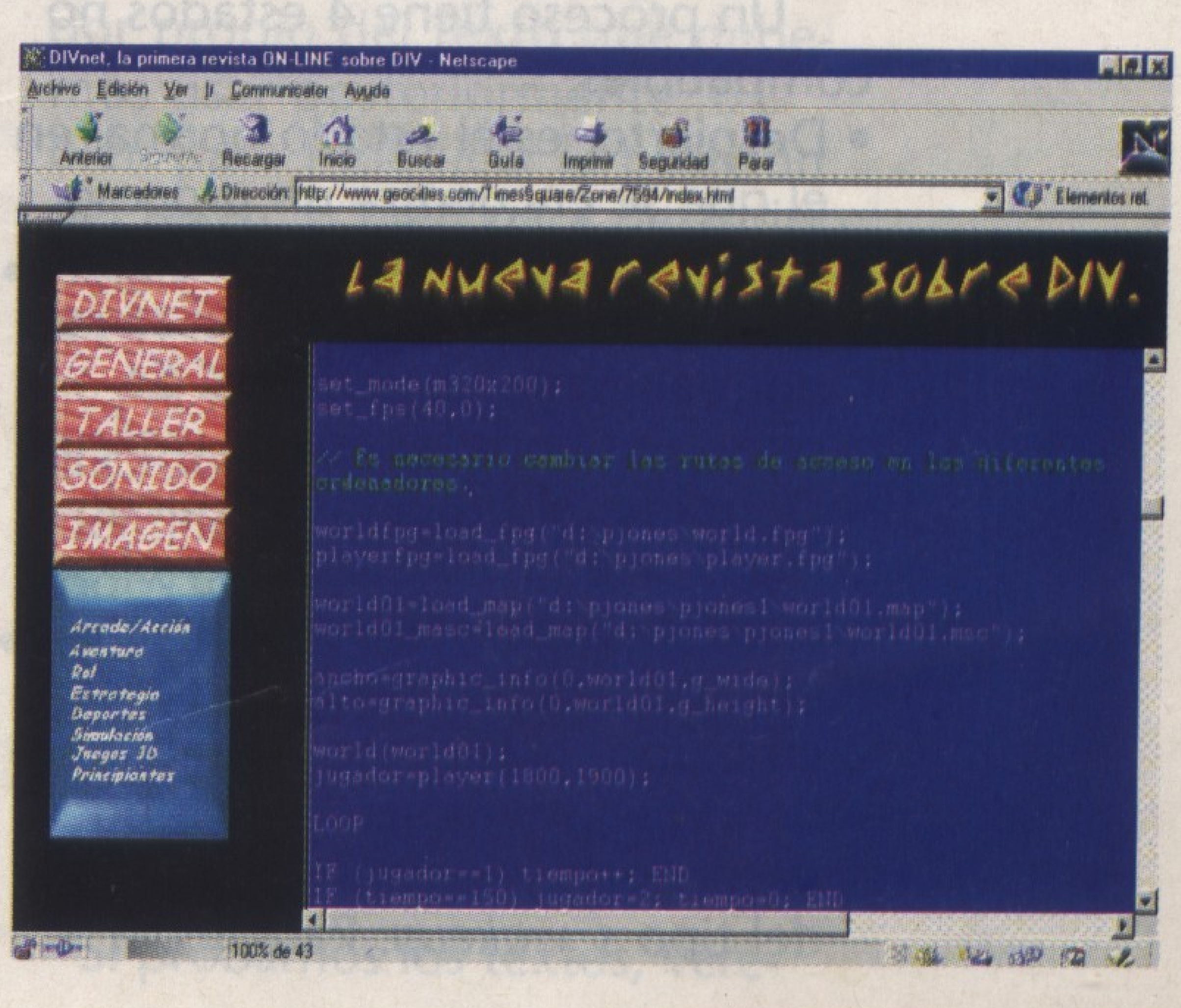
Alfredo del Barrio

Enlaces a visitar.

http://www.pagina.de/div2 (página de la revista DIVnet)
http://www.fortunecity.com/skyscraper/century/891 (página de Radikal-Div)
http://pagina.de/PapadaSoft (grupo PapadaSoft)
http://alpha2.dizan.informaticos.org (grupo Dizan de programación de videojuegos)
http://pagina.de/Benko
http://pagina.de/CrystalWave (grupo Crystal Wave)
http://geocities.com/TimesSquare/Galaxy/5141
http://divgames.com (página de DIV Games Studio)

DIVnet la primera revista electrónica en Internet dedicada al mundo DIV y DIV 2





Programando en DIV

Dando los últimos retoques

Todos los juegos precisan de otros factores que, aún no siendo imprescindibles, dotan a éstos de una mayor profesionalidad. Sonidos, intros y menús son los elementos que aprenderemos a desarrollar en DIV.

Una vez desarrollado el núcleo del juego, debemos comenzar a añadir esos retoques que servirán para hacer más vistoso nuestro juego. Sonidos, menús, música y presentación, entre otros, serán los que nos ocupen en este número.

Los menús y las señales

Todo juego puede contar con un menú principal donde poder configurarlo y modificarlo a nuestro antojo. La mejor forma de programar un menú, es considerando cada pantalla como un proceso independiente. De esta forma, cuando estemos en una pantalla este proceso permanecerá activo, mientras que el resto se encontrarán "dormidos". Cuando pasemos de una pantalla a otra, basta con despertar el proceso asignado a dicha pantalla y dormir el actual. Esto se hace con lo que se denomina "señales entre procesos".

> Cuando en un proceso se crean otros se produce una estructura jerarquizada similar al concepto de familia

Un proceso tiene 4 estados no compatibles:

- Despierto: es el estado normal en el que se encuentra un proceso en el momento de crearse. En él pueden ejecutarse y visualizarse normalmente las órdenes. Se representa por la señal *s_wakeup*.
- Dormido: el proceso no puede ejecutarse ni visualizarse, y el resto de procesos no detectan colisión alguna con éste, sin embargo, el proceso seguirá existiendo en memoria, pudiendo despertarlo mediante una señal del tipo *s_wakeup*. Para llegar a este estado, se debe utilizar la señal *s_sleep*.
- Congelado: el proceso no podrá ejecutarse, pero seguirá visualizándose, pudiendo colisionar con otros procesos. El objeto sigue por tanto existiendo en memoria. Para congelar un proceso debemos enviar la señal *s_freeze*.
- Muerto: se trata de un estado transitorio. El proceso deja de ejecutarse y visualizarse y se elimina de la memoria en la siguiente visualización o frame. Para matar a un proceso debemos enviarle la señal *s_kill*.

Cuando dentro de un proceso, se crean otros procesos, se produce una estructura jerarquizada similar al concepto de familia. Los procesos creados se llaman hijos, y el creador padre. De esta forma si los hijos crean nuevos procesos, serán nietos del padre de éstos. Pero, ¿para qué sirve en realidad todo este embrollo familiar?. Pues bien, DIV nos permite enviar los mensajes anteriormente citados a un proceso y a todos sus hijos. De esta forma, una pantalla de un menú o del propio juego, puede tener varios procesos creados, de manera que conociendo al padre o proceso de la pantalla y enviándole el mensaje adecuado, podemos matar, dormir o despertar a todos sus procesos hijos. Los mensajes que podemos enviar al proceso padre y a toda su descendencia son:

S_wakeup_tree
S_sleep_tree
S_freeze_tree
S_kill_tree

Además, cada proceso, sea cual sea, posee varias variables locales que contienen el identificador de su padre, de su hijo mayor (el primero que creó), el de sus hermanos mayor y menor, así como el suyo propio. Estas variables son *father* (identificador del padre), *son* (identificador del hijo), *bigbro* (identificador del hermano mayor o primer proceso creado por el padre), *smallbro* (identificador del hermano menor o último proceso creado por el padre) e *id* (identificador del propio proceso), que valdrán 0 si alguno de ellos no existe.

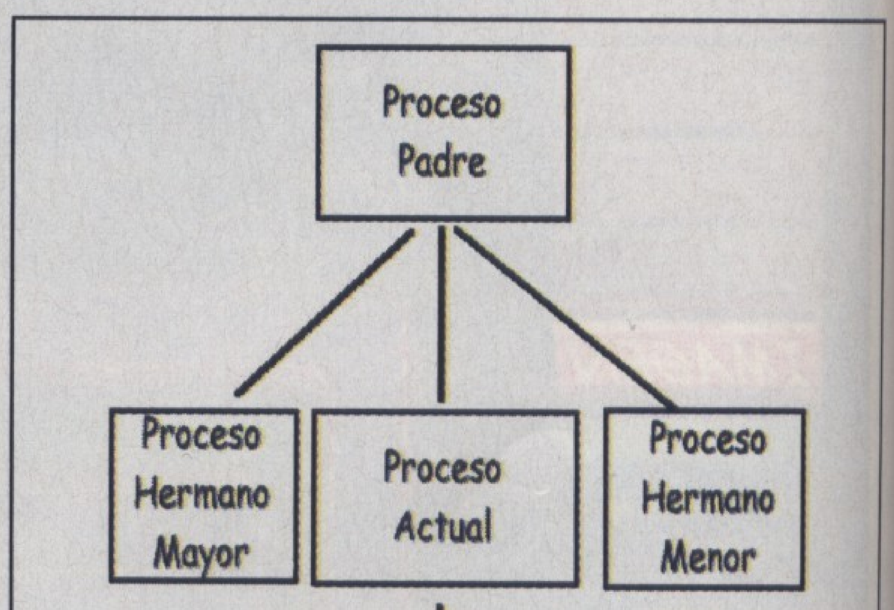
Una vez realizada toda esta introducción, veamos la instrucción que debemos escribir para enviar estos mensajes: Signal. Su estructura es:

Signal (<identificador de proceso>, <mensaje>);
Signal (TYPE <tipo de proceso>, <mensaje>);

La diferencia entre ambas es que la primera manda el mensaje sólo a un proceso, mientras que la segunda lo manda a todos los del mismo tipo.

Por tanto, veamos el código necesario para pasar de un menú a otro. Supongamos que estamos en el menú principal y pasamos al menú de opciones. Creamos por tanto un proceso opciones que contenga el código:

```
PROCESS opciones()
BEGIN
//...
signal (father,s_sleep_tree);
signal (id, s_wakeup);
LOOP
        //...
        FRAME;
END
```



Arbol de jerarquías entre procesos.

```
Signal (father, s_wakeup_tree);
Signal (id,s_kill_tree);
END
```

Al crearse, duerme todos los procesos creados por el padre, de forma que desaparezcan de la pantalla, pero se conserven en memoria, y como el proceso actual se encuentra entre ellos, se despierta a sí mismo. Cuando termina la ejecución del proceso opciones, lo matamos, así como a todos sus hijos, y despertamos al proceso padre y a todos sus procesos hijos, excepto al actual que ya está muerto.

Este es el concepto básico que se debe aplicar a todos los procesos. Ahora veamos cómo podemos diseñar los distintos elementos propios de un menú, como pueden ser los botones y los botones de varios estados, así como los textos.

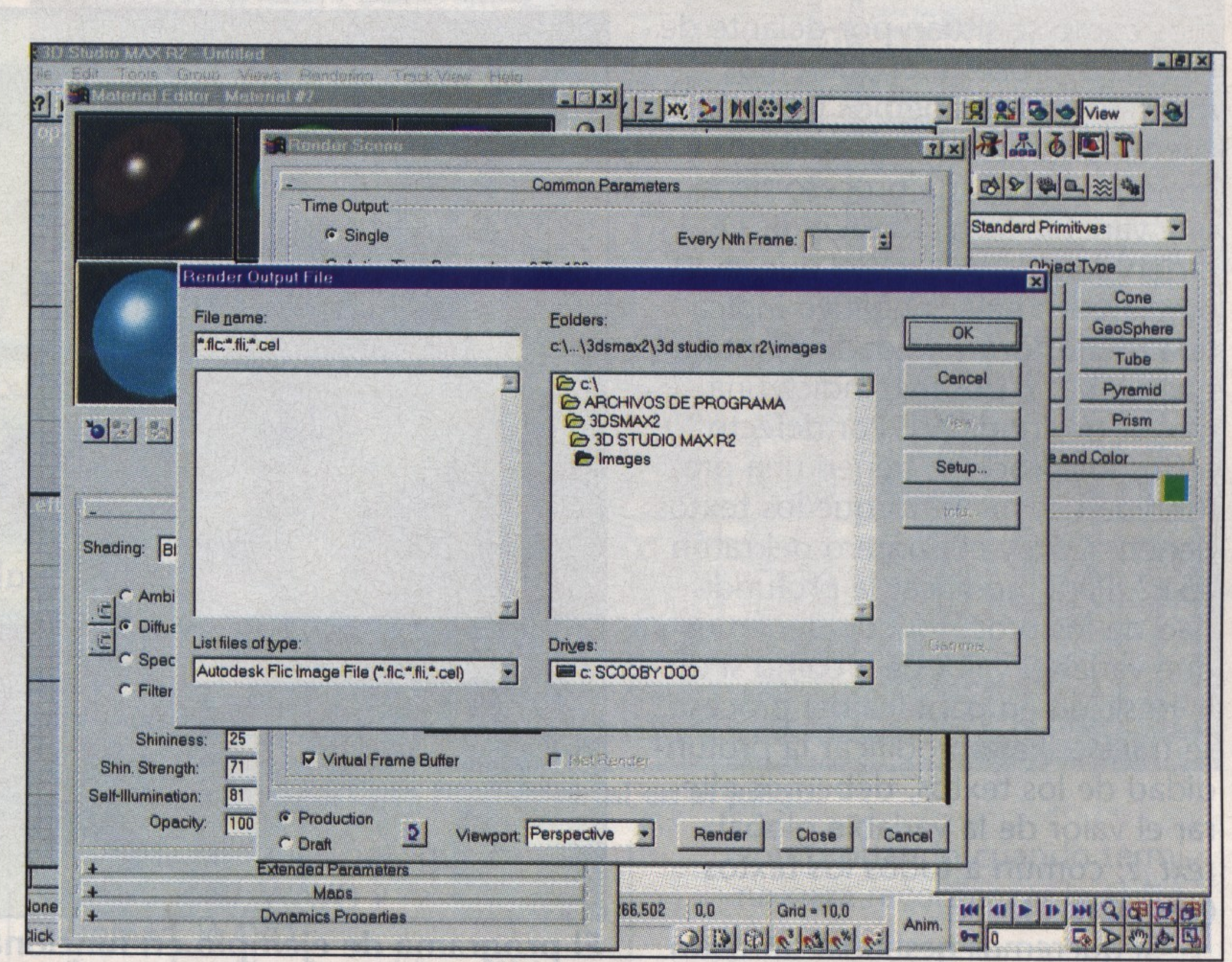
Controlando los menús: Botones

Cada día, pulsamos cientos de veces esos tan frecuentes botones de *Windows*. ¿Nunca os habéis preguntado cómo se hacen?. Pues es tan sencillo como disponer de dos mapas gráficos distintos, uno para cada estado: pulsado o sin pulsar. En consecuencia, debemos verificar el estado del ratón cada vez que se procese un *frame* para ver cual de los mapas mostramos por pantalla:

- Pulsado: si existe colisión entre el proceso y el ratón y está pulsado el botón izquierdo del ratón.
- Sin pulsar: cualquier otro caso.
- Traduciendo a código lo anterior y siendo los índices de los mapas 1 (sin pulsar) y 2 (pulsado):

```
PROCESS boton(x,y)
BEGIN
 Graph = 1;
 LOOP
  IF (collision(TYPE mouse) AND
mouse.left)
           Graph=2;
  ELSE
           Graph = 1;
  END
  FRAME;
 END
END
```

En cuanto a cómo crear los gráficos, decir que existen numerosos programas de retoque fotográfico que tienen efectos denominados *Buttonize* con los que realizar efectos de botón a cualquier gráfico. Además de este tipo de botones, también existen otros que cambian su imagen cuando el cursor se encuentra encima, con lo que para



Con 3Dstudio MAX podemos crear también animaciones FLI/FLC de forma sencilla.

programarlos debemos disponer de tres imágenes cuyos índices serán por ejemplo 1(inactivo), 2 (activo) y 3 (pulsado). Veamos el código necesario para estos botones, que como veréis será muy similar al anterior:

```
PROCESS boton2(x,y)
BEGIN
 Graph = 1;
LOOP
 IF (collision(TYPE mouse) )
          // si lo deseamos
podemos cambiar el cursor del ratón
mediante
          // mouse.graph = 4;
siendo 4 un nuevo tipo de cursor
  IF (mouse.left)
               Graph = 3;
  // actual en consecuencia a la
pulsación
   ELSE
               Graph = 2;
   END
 ELSE
          Graph = 1;
 END
 FRAME;
END
END
```

Hemos visto dos casos sencillos de cómo realizar botones. No obstante existen otras variaciones, pero con esta base no encontraréis dificultad alguna en cómo programarlos.

Escribiendo textos

Casi cualquier menú dispone de textos que nos indican las acciones que se pueden realizar. En el número anterior vimos cómo podíamos escribir números en la pantalla y que se actualizasen solos, así como la creación de nuevas fuentes de texto. Veamos ahora cómo se puede escribir cualquier tipo de texto.

En DIV, cuando creamos un texto, éste permanece en pantalla hasta que lo destruyamos. Permanecerá inalterado, y tan sólo podremos modificar su posición. Si queremos cambiar el texto, debemos destruirlo y crear uno nuevo con un texto distinto. Las funciones que se usan para ello son:

- Write (<fuente de texto>, <x>, <y>, <código de centrado>, <texto>): escribe el texto en la posición (x, y) de la pantalla, siendo dicha coordenada el punto indicado por el código de centrado. Para ver los posibles códigos de centrado, consulta la tabla 1. Debemos enviarle como primer parámetro el código de la fuente de texto deseada que hemos cargado previamente, ó 0 si deseamos usar la fuente del sistema. La función devuelve un identificador propio del texto que debemos usar posteriormente, o bien para modificar la posición del texto, o bien para destruirlo.
- Delete_text(<identificador de texto>): elimina de forma definitiva el texto indicado por el identificador. Si se le envía como identificador *all_text* se eliminan todos los textos.
- Move_text(<identificador de texto>, <x>, <y>): desplaza el texto indicado por el identificador a la posición (x,y) de la pantalla.

Si probamos los textos, vere-

mos cómo se sitúan por delante de cualquier proceso. Si queremos modificar esto debemos tener en cuenta primero el concepto de profundidad. Cada proceso tiene una variable local *z* que indica el orden de prioridad en el que se dibujan. Un valor negativo indica una menor profundidad, mientras que un valor positivo indica una mayor profundidad. Por defecto todos los procesos tienen una profundidad 0, mientras que los textos tienen –256 y el puntero del ratón –512. Para modificar la profundidad de los procesos sólo tenemos que variar el valor de z como si de la posición en pantalla del proceso se tratase. Para modificar la profundidad de los textos, debemos alterar el valor de la variable global *text_z*, común a todos los textos (ver tabla 1).

Si queremos gestionar un texto como si de un proceso se tratara, podemos crear un proceso sin una imagen asociada y realizar en él todas las operaciones que creamos convenientes.

> Un valor negativo indica menor profundidad

Veamos un ejemplo de un texto que se desliza hasta que desaparece por la derecha:

```
PROCESS textodeslizante()
PRIVATE
Ident;
Fuente;
BEGIN
X=0;
Y=0;
Fuente = Load_fnt("fuente.fnt");
Ident = write (fuente,x,y,0,"Texto deslizante");
WHILE (x < 640) // ancho de la pantalla
  X+=2;
  Move_text(ident, x,y);
  FRAME;
END
Delete_text(ident);
END
```

La presentación o intro

Todo juego que se precie tiene una presentación o vídeo de introducción. DIV no es ajeno a ello y nos permite cargar animaciones externas en formato FLI/FLC de *Autodesk*. Existen en el mercado innumerables programas que generan este tipo de secuencias, como puede ser la serie *3Dstudio*. En el programa de ejemplo que acompaña al CD hemos incluido una animación de ejemplo que introduce al programa del número anterior.

Para mostrar una animación debemos, en primer lugar, verificar que la resolución de la pantalla nos permite mostrar la imagen sin peligro de rebasar los bordes. Una vez realizada la comprobación (basta con poner el modo de pantalla adecuado para el tamaño de la animación), debemos cargar la animación. En este momento, y habiendo llegado a la sentencia *FRAME*, se mostrará el primer frame de la animación. Si no realizamos ninguna acción más, se repetirá indefinidamente este frame. Por ello, debemos utilizar una función *frame_fli()* para que pase al siguiente frame. Una vez terminada, debemos eliminar la animación para que desaparezca definitivamente de la pantalla. Veamos las instrucciones de que disponemos para ello:

- Start_fli(<nombre del archivo>,<x>,<y>): carga una secuencia FLI/FLC para mostrarla en la posición (x,y) de la pantalla. Sólo se puede tener una animación activa al mismo tiempo.
- Frame_fli(): avanza al siguiente frame de la animación. Si ha llegado al final de ésta, devuelve falso (0), y si no, verdadero.

  Reset_fli(): hace retornar al primer frame de la animación. Permite mostrar de forma cíclica una animación.
- End_fli(): descarga de memoria la animación cargada previamente con Strat_fli().

Veamos un ejemplo de cómo podemos reproducir una animación en DIV:

```
PROGRAM coches;
BEGIN
  // Se cargan y se ejecutan las distintas animaciones iniciales
  set_mode(m320x200);
  start_fli("c:\div\divmania\coches.fli",0,0);
    WHILE (frame_fli()<>0 AND NOT mouse.left) FRAME; END
    end_fli();
    LOOP
  // aquí viene el programa principal
       FRAME;
  END
  END
```

Si además de con el ratón, queremos interrumpir la animación cuando se pulse una tecla, debemos recurrir a una variable global denominada *scan_code*. Su valor es 0 si en el frame anterior



El programa de ejemplo en funcionamiento.

Tabla 1. Código de centrado de los textos

Código de centrado	Significado
0	Arriba izquierda
1	Arriba
2	Arriba derecha
3	Izquierda
4	Centro
5	Derecha
6	Abajo izquierda
7	Abajo
8	Abajo derecha

no se ha pulsado ninguna tecla. Por tanto si queremos verificar la pulsación de alguna tecla, debemos añadir la secuencia:

```
WHILE (frame_fli()<>0 AND
scan_code==0 AND NOT mouse.left)
```

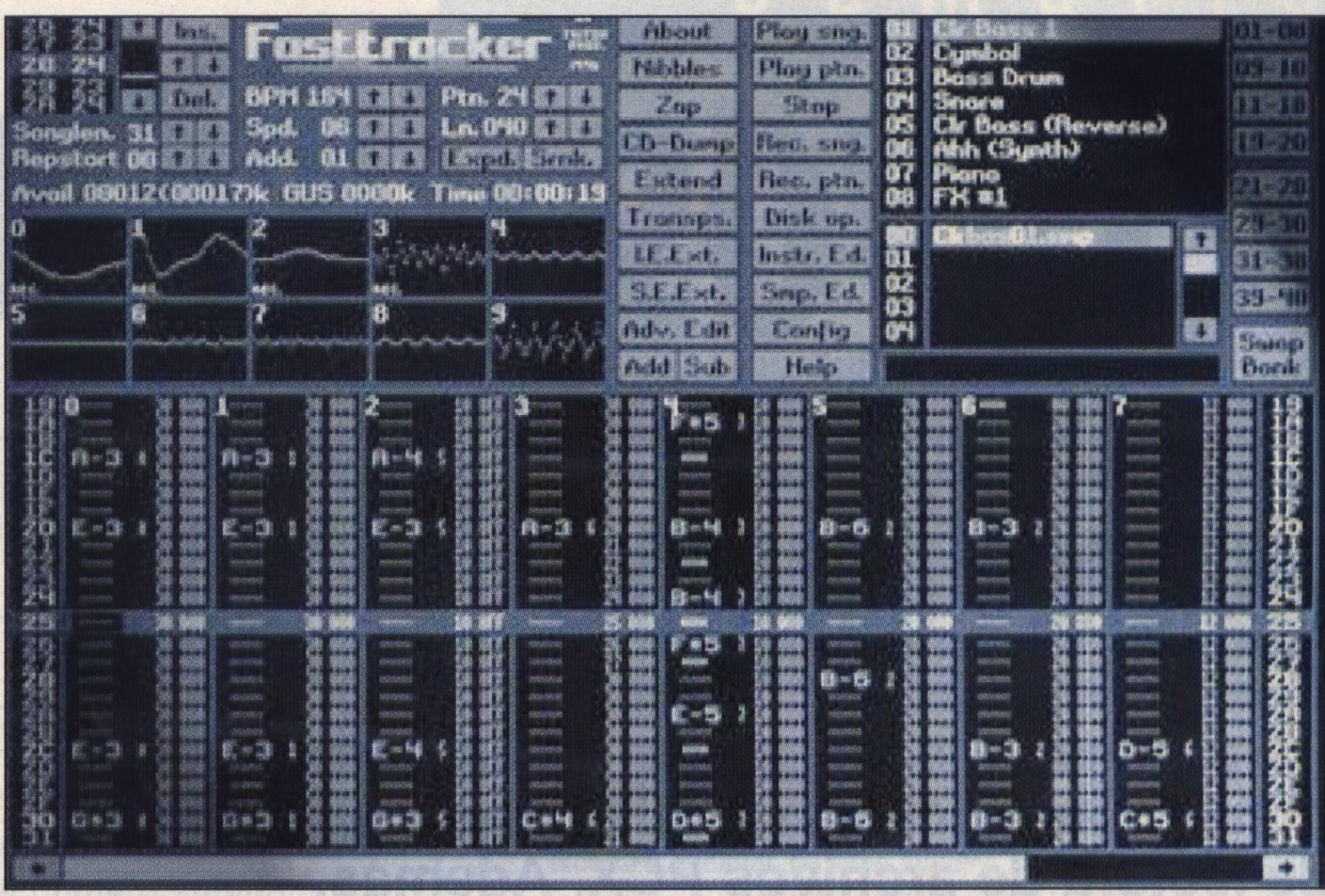
Para eliminar posibles pulsaciones de teclas en instantes anteriores a la apertura de la animación, debemos asignar manualmente el valor nulo a la variable *scan_code*.

Hasta aquí la reproducción de animaciones, ahora tan sólo nos falta la reproducción de sonido para terminar con nuestros retoques al juego del anterior número.

Reproduciendo sonidos.

Los sonidos pueden ser tan importantes como lo son unas buenas imágenes. Aporta el realismo necesario para complementarse con las imágenes y generar un escenario casi real. Por ello vamos a estudiar las principales funciones que nos permiten reproducir sonidos. Hay que notar que también incluiremos funciones específicas de *DIV 2*, como son las propias de reproducción de archivos WAV y canciones XM, S3M y MOD:

- Reproducción de archivos de sonido PCM: disponemos de 4 funciones para manejar sonidos:
  - Load_PCM(<nombre de archivo>,<cíclico>): carga el archivo indicado, devolviendo un código identificador que debemos utilizar más tarde para reproducirlo. Si queremos que se reproduzca de forma cíclica, debemos poner a 1 el argumento cíclico, y si no a 0.
  - Sound (<código>,<volumen>,<frecuencia>): reproduce el sonido cuyo código indicamos con el volumen y frecuencia determinados. Ambos valores varían entre 0 y 256. Si queremos que la frecuencia sea la misma que la propia del sonido debemos especificar 256 en el valor frecuencia. La función devuelve el canal de sonido por el que se reproduce, valor que debemos guardar si queremos parar la reproducción.
  - Stop_sound(<canal>): detiene la reproducción del sonido que se está reproduciendo por el canal especificado, devuelto por la función sound.
  - Unload_PCM(<código>): elimina de la memoria el archivo de sonido cuyo código se le envía.
- Reproducción de archivos de sonido WAV: análogo a los archivos PCM, excepto en los nombres de las funciones, que cambian a load_WAV() y Unload_WAV().
- Reproducción de archivos XM, S3M y MOD: su control es mucho más sencillo que los anteriores. Tan sólo se puede reproducir una canción de este tipo al mismo tiempo. Las funciones para gestionarlo son:
  - Load_song(<nombre del fichero>,<cíclico>): carga el fichero musical con el nombre indicado. Si deseamos que se reproduzca de forma cíclica enviamos en el segundo parámetro un 1, si no un 0. La función devuelve un identificador necesario para posteriores acciones.
  - Song(<identificador>): reproduce la canción con el identificador indicado.
  - Stop_song(): detiene la reproducción de la canción que se está ejecutando. Como sólo se puede reproducir una a la vez, no necesita parámetros.
  - Unload_song(<identificador>): descarga el fichero musical cargado previamente y libera la memoria ocupada por este.
- Reproducción de Audio CD: para reproducir pistas de audio de un CD. Sus funciones son:
  - Play_CD(<número de pista>, <modo>): reproduce la pista de audio indicada. El modo indica si queremos tocar la canción y pararnos (valor 0) o seguir con las siguientes (valor 1).
  - Stop_CD(): detiene la reproducción del CD de audio.
  - Is_playing_CD(): devuelve cierto si está sonando el CD, y falso si no. Podemos usarla para reproducir indefinidamente una pista de audio indicando en el modo de reproducción el valor 0 y comprobando cuando termina para recomenzar la reproducción.



El sonido es un apartado fundamental en el juego.

Veamos como ejemplo de su uso, el código para reproducir una canción XM y una misma pista de audio de forma repetida:

```
PROGRAM sonido;
GLOBAL
    Identificador;
BEGIN
    Identificador =
load_song("archivo.xm",1);
    Song(Identificador);
    LOOP
        PlayCD(4);
        //... aquí va el resto
del programa
        IF (key(_f1))
stop_song(); END
        FRAME;
    END
    Unload_song(identificador);
END

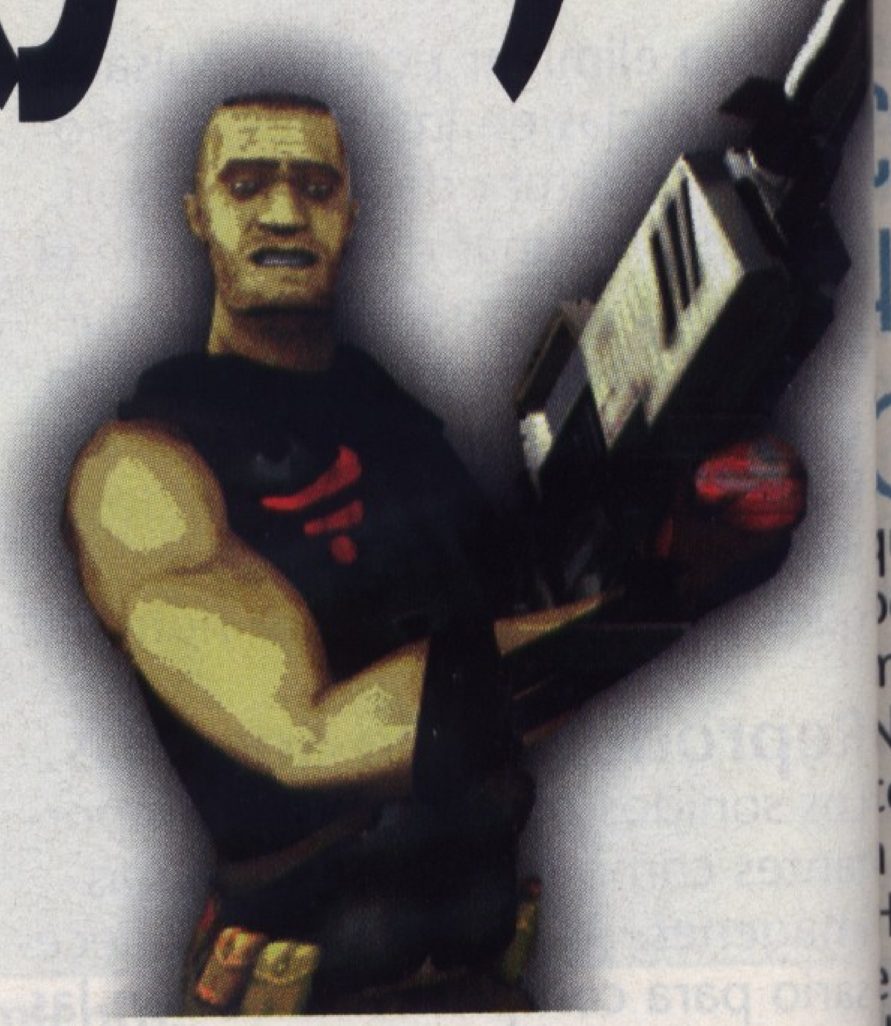
PROCESS PlayCD(pista)
BEGIN
    Play_CD(pista,0);
    LOOP
        IF
(is_playing_CD()==0)
Play_CD(pista,0);
        END
        FRAME;
    END
END
```

Esto es todo por hoy, y no es poco. Aconsejo que experimentes con todo lo que has aprendido y sobre todo, consultes el código fuente de los programas de ejemplo que vienen con la versión comercial de DIV. Para cualquier consulta, crítica u opinión, puedes enviar un e-mail a trinidad@arrakis.es

Pablo Trinidad

Archivos PCX (y II)

Manipulando imágenes



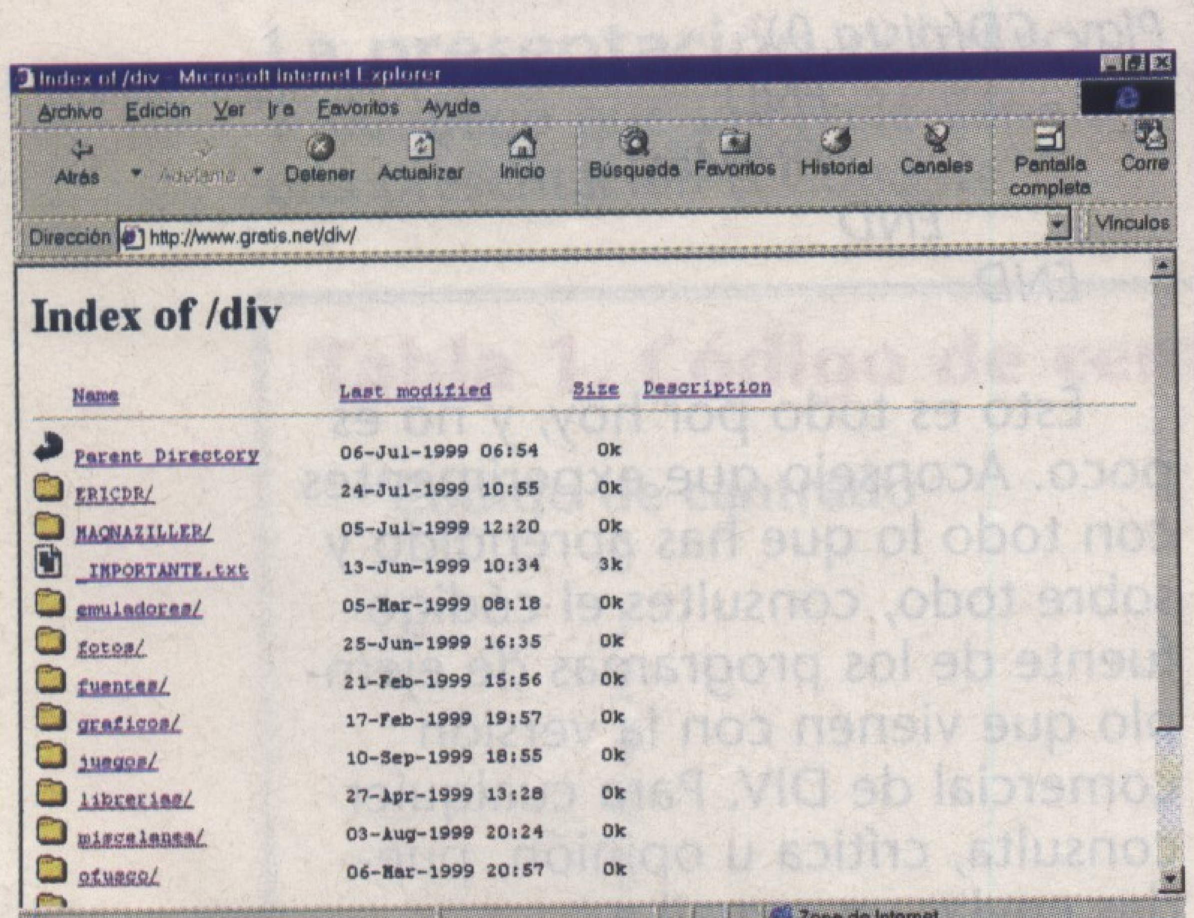
Para rematar la faena que comenzamos en el número anterior, desarrollaremos la función de lectura de archivos PCX. Además, descubriremos cómo compilar nuestras DLLs con Borland C++.

Para muchos usuarios de *DIV* ha supuesto un obstáculo para el desarrollo de bibliotecas o *DLLs* el que, por construcción, esté limitado a *Watcom C++*. Por eso, y gracias a la colaboración de José M. Sevillano, podemos ahora crear nuestras bibliotecas con *Borland C++*, además de abrir un camino importante hacia la investigación en otros entornos. Veamos en qué consiste este gran "descubrimiento".

El problema de la compilación. Nombres de funciones

Una DLL en *Windows 95/98/NT* (las usadas por *DIV*) no es más que una aglomeración de funciones que podemos usar de forma dinámica en una aplicación.

DIV es un programa de *DOS*, que trabaja con lo que se denomina DPMI, es decir, un gestor de modo protegido de 32 bits (386 ó superior). Por ello, necesita de DLLs también de 32 bits, como son las de *Windows 95/98/NT*.



En el FTP de DIV (www.gratis.net/div) podemos encontrar gran cantidad de librerías así como muchas otras cosas útiles.

Dichas bibliotecas, de enlace dinámico como se denominan, contienen una serie de funciones que se encuentran cada una en un punto del fichero, en lo que se llama punto de entrada (*entry-points*). Las aplicaciones de *Windows*, acceden a ellas a través de estos puntos de entrada, que debemos conocer de antemano e indicárselos al compilador. Sin embargo, *DIV* accede a las funciones a través de su nombre, buscando su punto de entrada para así poder ejecutarlas.

Cada entorno codifica los nombres de las funciones y el número y tipo de parámetros que recibe de distinta manera, de forma que para buscar una función por su nombre debemos tener en cuenta de antemano el compilador en el que fue desarrollada. Sin embargo, *DIV* tan sólo reconoce el formato de codificación de funciones de *Watcom C++*, ya que para acceder a las funciones no usa los puntos de entrada. Por tanto, debemos de alguna manera hacer que el compilador cambie esta denominación. Pongamos un ejemplo para poder imaginarnos este concepto: la función *divmain* se representa en *Borland C++* como *@divmain$qpqpc$pvpqpcpv$v*. Rebuscado, ¿verdad? Pues bien, lo que debemos hacer entonces será indicarle al compilador que nos exporte todas las funciones con otro nombre. Esto se hace en lo que se denomina archivo de definición de módulos (DEF). Pero, ¿con qué nombre? Pues aquí entra en acción *DIV*. El programa *DIV* reconoce también como nombre el de la función con un carácter de subrayado como prefijo o sufijo, es decir, que un nombre válido para la función *divmain* sería *_divmain o divmain_*. Por tanto, si le indicamos en dicho archivo DEF que nos asocie los nombres de *divmain*, *divlibrary* y *divend* a sus correspondientes en *Borland* o en cualquier otro entorno, no habrá problema con que *DIV* no detecte los puntos de entrada de las funciones.

Es importante saber que a lo sumo se exportarán sólo estas 3 funciones, y que el resto de funciones creadas por nosotros están contenidas por construcción dentro de ellas, con lo que no hará falta buscar el nombre de cada nueva función y añadirlo al archivo de definición.

El problema de los parámetros

Pero todo no acaba aquí. También existe un problema con los parámetros enviados a las funciones. Hay varias formas de enviar parámetros a una función:

Mediante la pila: es el método más usado. Los parámetros se depositan por orden en la pila del sistema para ser recogidos dentro del código de la función en orden inverso. Pero hay dos formas de depositar los datos en la pila:

- Pascal: deposita los datos de izquierda a derecha. Se representa por __pascal en el lenguaje C.
- C: deposita los datos de derecha a izquierda. Se representa por __cdecl.
- Standard call: es el método utilizado por la API 32 de Windows. Básicamente es igual al tipo __cdecl, con la excepción que no necesita mantenimiento de pila.

Fastcall: es un nuevo tipo de llamada que utiliza además de la pila, algunos registros de 32 bits del procesador. No hay un tipo estándar, con lo que varía según el compilador:

DIV developer

NÚMERO 4

Sumario

Curso de programación y concurso

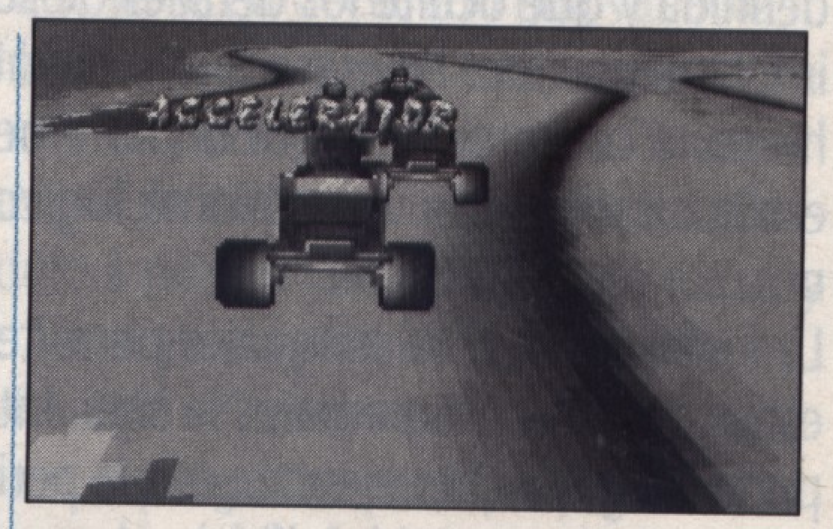
Comienza el curso en todos los colegios españoles, así que es una buena época para poner al día nuestros conocimientos de programación. Nuestros cursos están diseñados con la única finalidad de enseñar a programar a nuestros lectores. Hemos podido comprobar que esta sección es una de las preferidas de nuestros lectores. Muchos de ellos se han iniciado en el mundo de los videojuegos gracias a DIV y ahora quieren ampliar sus horizontes; otros ya tenían conocimientos previos y quieren saber más, y, cómo no, también los habrá que, siendo profesionales, quieran echar un vistazo a los conceptos básicos. Esperamos que más de uno termine realizando sus propios juegos y nos los envíe para participar en el concurso. Y que se haga realidad la frase "muchos serán los llamados y pocos los elegidos".

Continuamos con los siguientes capítulos iniciados en el número anterior de nuestra revista. Como ya sabéis, nuestro primer curso trata de introducir a la programación a los no iniciados. Para ello continúa su repaso a la programación básica, es decir la programación basada en algoritmos. El segundo curso también sigue la línea del número anterior introduciéndonos a la programación en C, un lenguaje de programación que ya es algo más que un clásico, aunque mantiene su vigor gracias a sus excepcionales características.
Por último, tenemos el curso de programación en ensamblador donde se continúa dando cuenta de todo lo que tiene que ver con el lenguaje que utilizan los microprocesadores de nuestros ordenadores. Y como no, os damos cuenta de los tres títulos que han resultado elegidos en nuestro concurso de programación con DIV. En nuestras páginas encontraréis cumplida información sobre ellos, con especial atención lógicamente a los aspectos de programación de los tres juegos.



Por supuesto, nuestro concurso sigue en pie con los mismos premios. Ya sabéis: 25.000 pesetas para el ganador y 20.000 para los otros dos juegos elegidos. Esperamos con impaciencia recibir vuestros juegos. Ya sabéis dónde mandarlos, pero por si acaso hay algún despistado, os recordamos que podéis hacerlo por e-mail o correo normal:

Nuestra dirección:
gover@prensatecnica.com



Divmanía
C/ Alfonso Gómez, 42
Nave 1-1-2
28037, Madrid, España



Destacamos

En nuestro CD de portada incluimos los juegos que han resultado ganadores en este número y, cómo no, sus respectivos códigos para que veáis todo el proceso de creación. Estos son los afortunados:

- Mr. Bones (o vete al infierno), un excelente juego de plataformas.
- Acelerator, para que seas el más rápido del circuito.
- E. J. Dominó, para que jugar al dominó contra el ordenador.

Módulos

Gracias a que los lenguajes de programación permiten la definición de módulos, es posible dotar a los programas de una estructura comprensible y manejable para el programador, además de facilitar la reutilización del software.

Aunque el conjunto de instrucciones descrito en el anterior artículo ayuda a la estructuración de un programa evitando la utilización de la instrucción de salto incondicional, no son éstas ni mucho menos las únicas estructuras que ayudan por sí mismas a la estructuración de un programa. Se puede decir que la estructuración consiste en construir funciones o módulos de un nivel más alto que las que posee un lenguaje de programación, con un interfaz claramente definida y que oculte los detalles de su implementación. Piénsese que la mente humana comprende mejor un fenómeno estructurado y con un esquema lógico, que otro sin estos atributos.

La forma en que los lenguajes permiten estructurar un programa es la siguiente: en primer lugar se tiene en cuenta la pericia del programador para identificar una secuencia de instrucciones que en conjunto realizan una tarea determinada. A continuación, el programador debe seleccionar la forma en que dicha secuencia de instrucciones se comunica en su conjunto con el resto del programa (interfaz) y por último se le dota a dicha secuencia de instrucciones de un nombre común, de tal forma que puede ser utilizada en su totalidad a través de dicho nombre.

En la figura 1 se encuentran dos secuencias de programa equivalentes, en lo que se refiere a los resultados, que reflejan las ideas anteriormente descritas. La primera secuencia

```
Program areas ;
const
 pi= 3.1416 ;
var
 longitud : real ;
begin
 clrscr ; {borra la pantalla}
 {validación de la entrada}
 repeat
  write ('Introduzca una longitud') ;
  readln(longitud) ;
 until longitud > 0 ;
 {cálculo del área del círculo}
 area := pi*longitud^2 ;
 writeln ('El área del círculo es ',
        area) ;
 {cálculo del área del cuadrado}
 area := longitud*longitud ;
 writeln ('El área del cuadrado es ',
        area) ;
end.{fin del programa}
```

```
Program areas ;
const
 pi= 3.1416 ;
var
 longitud : real ;
procedure valida (l : real) ;
begin
 repeat
  write ('Introduzca una longitud') ;
  readln(l) ;
 until l > 0 ;
end ;
function area_circulo (radio : real) :
         real ;
begin
 area_circulo := pi*radio^2 ;
end ;

function area_cuadrado (lado : real) :
         real ;
begin
  area_cuadrado := lado*lado ;
end ;
 clrscr ;
 valida (longitud) ;
 writeln ('El área del círculo es ',
         area_circulo(longitud)) ;
 writeln ('El área del cuadrado es ',
         area_cuadrado(longitud)) ;
end.{fin del programa}
```

FIGURA 1.

de programa permite aceptar desde el teclado un número mayor que cero, que servirá para calcular el área del cuadrado que tiene dicho número por lado y el área del círculo que tiene por radio el citado dígito. Como se puede comprobar, la validación y el cálculo de las correspondientes áreas están mezcladas en una única secuencia de instrucciones, de tal forma que no se distinguen unas funciones de otras. En la segunda secuencia se han construido determinadas funciones y procedimientos (módulos), con el objeto de distinguir las distintas funciones que se realizan dentro del programa. Se puede observar, entre otras cualidades, la claridad de la segunda solución frente a la primera, debido a que las funciones independientes se han agrupado en módulos.

PARAMETROS

En la declaración de un módulo se encuentra una lista de parámetros formales. Éstos constituyen el interfaz (o al menos lo deberían formar), en compañía de otros módulos o el propio programa. Cada uno de dichos parámetros son locales a los módulos, es decir, que pueden ser referenciados a través de su identificador desde dentro del módulo.

Respecto a cómo se comportan los parámetros en relación al módulo, pueden distinguirse dos tipos de parámetros: por valor y por dirección. Los parámetros formales pasados por valor actúan como una variable local del módulo, excepto cuando toman su valor inicial del valor correspondiente del parámetro actual en el momento de la llamada al módulo. Hay que señalar que los cambios producidos en el parámetro formal no afectan al parámetro actual, por lo que al finalizar la ejecución del módulo aquel tendrá el mismo valor que antes de su llamada. Esto sucederá incluso aunque dentro del módulo se encuentre una instrucción de asignación a dicho parámetro.

El valor admisible de un parámetro por valor es una expresión cuya evaluación debe ser del mismo tipo que la del parámetro formal correspondiente a la cabecera del módulo. Un parámetro por valor en Pascal puede distinguirse por no ir precedido en la cabecera del módulo por la palabra reservada **var**.

Los parámetros formales por dirección únicamente pueden ser sustituidos por variables, debido a que dentro del módulo cualquier asignación hacia el parámetro formal implicará un cambio en el parámetro actual. Por lo tanto, cuando un módulo finalice su ejecución al ser llamado, puede ocurrir que los parámetros actuales correspondientes a los de tipo formal en la llamada al módulo sean modificados.

Cuando en la cabecera de un módulo en Pascal un parámetro formal viene precedido por la palabra reservada **var**, se trata de un parámetro por dirección. De lo descrito anteriormente se puede concluir que se utilizarán parámetros por valor cuando el módulo tan sólo necesite conocer el valor del parámetro pero no tenga intención de modificarlo. Si se desea que un parámetro sufra algún cambio, hay que pasar aquel por dirección. Además, es necesario tener en cuenta en las llamadas a los diferentes módulos que el orden de los parámetros resulta muy significativo a la hora de hacer equivalentes los diferentes parámetros formales (aquellos que se encuentran en la definición del módulo) y los denominados parámetros actuales (aquellos que se utilizan al realizar la llamada al módulo).

Los lenguajes estructuran los programas mediante secuencias de instrucciones que el usuario ha de dominar

En la figura 2, ver página siguiente, se encuentra un programa con diversos módulos, que permitirá comprender mejor la descripción realizada previamente. Antes de comenzar el cuerpo del mismo se han declarado tres módulos correspondientes al procedimiento ***tabla_multiplicar***, que dispone de un parámetro formal por valor. Éste no se modificará a lo largo del procedimiento, a diferencia de lo que ocurre con el llamado de ***intercambio***, que dispone de dos parámetros formales por dirección. Dichas medidas sí se modificarán, pues el procedimiento debe intercambiar sus valores. La función ***mcd*** calcula el máximo común divisor dados dos números, al tiempo que ambos parámetros son pasados por valor. En este caso no se modificarán, aun cuando son la base del cálculo del máximo común divisor.

En el cuerpo del programa principal pueden verse las diferentes llamadas a los módulos. La llamada a los procedimientos se realiza disponiendo el nombre y entre paréntesis la denominación de los parámetros actuales. El

procedimiento sustituirá los parámetros formales por los de tipo actual, para pasar a ejecutar de inmediato sus instrucciones. Obsérvese la llamada al procedimiento ***tabla_multiplicar***. Sucede en este punto que el parámetro actual de la llamada es ***i*** y que cuando se ejecute el procedimiento, el parámetro formal ***número*** será sustituido por el parámetro actual. A continuación se ejecuta el conjunto de instrucciones asociadas al procedimiento, de tal forma que cuando finalice, el control se devuelve a la instrucción que sigue a la llamada al procedimiento. Hay que hacer notar también que el parámetro actual no sufrirá ninguna modificación, por lo que la variable *i* del programa principal valdrá exactamente lo mismo que cuando se produjo la llamada al procedimiento. Esto no ocurre cuando se efectúa la llamada al procedimiento ***intercambiar***, debido a que los parámetros que presenta éste son parámetros por dirección. Debido a ello, la llamada a dicho procedimiento con los parámetros actuales ***prim*** y ***seg*** modificará su contenido. La diferencia entre el paso de parámetros por valor y por dirección no es tan explícita en otros lenguajes de programación a la hora de declarar el tipo de parámetros, dándose la posibilidad de que sólo exista el paso por dirección.

Módulos

Un módulo no es más que una unidad definida y manejable que presenta un interfaz claramente definido, que puede compilarse de manera independiente y que posee una serie de instrucciones agrupadas por delimitadores, con un identificador global a través del cual se referencia. Además, realiza una función separable del programa que lo utiliza. En cuanto a la longitud de un módulo, no existe ninguna restricción, siendo un buen criterio no sobrepasar el tamaño de una página.

Puesto que la anterior definición es abstracta, se recurre a describir cómo se pueden definir y utilizar módulos en lenguaje Pascal. En otros lenguajes el concepto es similar, variando únicamente la forma en que se declaran los diferentes módulos.

Los módulos existentes en Pascal son los procedimientos y las funciones. Éstos permiten anidar bloques dentro del bloque del programa principal. Cada declaración de procedimiento o función dispone de una cabecera, seguida por un bloque de declaraciones e instrucciones. La llamada a un procedimiento se produce cuando en el cuerpo de un programa aparece el nombre del procedimiento, mientras que el de una función debe estar inmerso en una expresión.

La cabecera de un procedimiento se compone de la palabra reservada procedure, seguida de un identificador de procedimiento. A continuación y entre paréntesis, se encuentra la lista de parámetros formales, que da paso a una cabecera en la que se localiza el bloque asociado al procedimiento (en la figura 1 puede verse la declaración valida del procedimiento). De forma similar a la declaración de un procedimiento es posible declarar también una función. Las tres únicas diferencias se encuentran en que la cabecera de una función comienza con la palabra reservada function. Por otra parte, al final de la cabecera se debe indicar el tipo de valor que devolverá la función y la necesidad de asignar el valor devuelto mediante una asignación al nombre de la función (equivale a la sentencia return de otros lenguajes). Más adelante se ubica el bloque de instrucciones que pertenecen a la función. En la figura 1 se pueden ver las declaraciones de las funciones área_circulo y área_cuadrado, que en ambos casos devuelven un valor de tipo real.

```
Program varios ;
var
 i : integer ;
 prim, seg : integer ;
procedure intercambio (var x, y :
          integer) ;
 var
  auxiliar : integer ;
begin
  auxiliar := x ;
  x := y ;
  y := auxiliar ;
end ;
procedure tabla_multiplicar (numero :
          integer) ;
 i : integer ;
begin
 for i := 1 to 10
    do
     writeln (numero, ` por `, i, ` es
          `, numero*i) ;
end ;

 function mcd (dividendo, divisor :
          integer) : integer ;
  var
   resto : integer ;
 begin
  resto := dividendo mod divisor ;
  while resto <> 0
     do
      begin
        dividendo := divisor ;
        divisor := resto ;
        resto := dividendo mod divisor
      end ;
  mcd := divisor ;
 end ;

Begin {comienza el cuerpo del
        programa}
for i := 1 to 10
   do
    tabla_multiplicar (i) ;
 prim := 355 ;
 seg := 45 ;
 intercambiar (prim, seg) ;
 writeln (prim, seg) ;
 writeln (mcd (prim, seg) ;
end. {fin del programa}
```

FIGURA 2.

AMBITO DE LAS VARIABLES

El número de lugares donde resulta posible la declaración de variables, dentro de un programa, se ha ampliado al introducir el concepto de módulo, pues es posible declararlas tanto en el programa principal como en los diferentes módulos. Esta situación plantea dos problemas: por un lado, identificar los lugares donde se podrán utilizar las variables declaradas (ámbito), y por otro, las reglas a seguir para identificar la variable a usar cuando se ha seleccionado un mismo identificador para designar a variables en diferentes módulos (visibilidad).

Las variables declaradas en el programa principal se denominan globales y pueden ser utilizadas tanto en el cuerpo principal del programa como en cualquier módulo que se encuentre dentro del mismo. Las variables declaradas en un módulo se denominan variables locales y su ámbito abarca el módulo en el que han sido declaradas.

Si se utiliza el mismo identificador para declarar dos variables, una en el cuerpo principal del programa y otra en un módulo, la visibilidad de la variable global se pierde en éste. Todo el proceso implica que cuando se referencia el identificador de dicha variable en el módulo, se utilizará la declarada en el módulo y no la variable global. Dicha situación se complica cuando es posible construir módulos dentro de otros módulos (lo que ocurre en Pascal y otros lenguajes). La regla a seguir entonces es la misma que en el anterior caso. Por eso, la declaración de la variable del módulo más interno ocultará la declaración de la variable del más externo, mientras que en cuanto al ámbito, las variables declaradas en el módulo más externo serán visibles en aquellos otros más internos, salvo que éstos declaren variables con igual identificador que los módulos más externos.

En la figura 3 se puede observar un esquema de declaraciones que permitirá comprender mejor el concepto de ámbito y visibilidad. Se trata de las variables declaradas en el programa principal, visibles tanto en el módulo 1 como en el 2, salvo para las siguientes excepciones: en el primer módulo no se podrá ver la variable ***v01***, debido a que ya se ha declarado una variable con igual identificador que en éste; respecto al segundo módulo, no será visible por el mismo motivo la variable ***v02***. Las variables declaradas en el módulo 1 no son visibles en el número 2 y viceversa. Así, si en el módulo 2 se utiliza la variable ***v01***, ésta se referirá a la variable global declarada en el programa principal y no a la registrada en el módulo 1.

Como se comentará más adelante, se trata de una costumbre muy buena cuando se pretende programar y evitar al mismo tiempo la referencia directa a variables globales dentro de un módulo. La comunicación tanto entre éstos como a su vez entre el programa principal y los módulos debería realizarse a través del paso de parámetros.

```
Programa principal
  declaración de variables :
   v01, v02, v03
  Módulo 1
  ↑declaración de variables :
  |  v01, L11, L12                  Son visibles
  |                                 las variables
  |                                 globales
  ↓                                 v02,v03 y las
  Fin del Módulo 1                  locales L11 y
                                    L12.
  Módulo 2
  ↑declaración de variables :
  |  v02, L21, L22                  Son visibles
  |                                 las variables
  |                                 globales v01,
  ↓                                 v03 y las
  Fin del Módulo 2                  locales L21 y
                                    L22
Inicio Programa Principal
↑
|                                   Solamente son
↓                                   visibles las
Fin del Programa Principal          variables
                                    globales.
```

FIGURA 3.

Instrucciones condicionales

En este capítulo del curso se va a explicar otro de los temas más importantes de la programación en general: las famosas instrucciones condicionales, gracias a las cuales, podremos crear programas que ejecutarán, según sean las condiciones que nosotros mismos indiquemos, un código u otro.

Una vez visto el capítulo anterior todo lo referente a bucles, ahora es el mejor momento para pasar a explicar otro de los temas más importantes de la programación, no sólo ya del lenguaje C, sino de cualquier lenguaje: las instrucciones condicionales. Las instrucciones condicionales permiten crear aplicaciones que no ejecuten siempre el mismo código, sino que ejecutarán unas partes u otras del propio código de programa según se vayan dando unas condiciones u otras. A continuación se detalla todo el tema.

LA INSTRUCCION 'IF'

Como en casi todos los lenguajes de programación que existen (incluyendo al BASIC, PASCAL y otros muy conocidos) el C usa la palabra *if* como nombre de instrucción básica condicional. Veamos un ejemplo de su uso:

```
/* Uso de IF */
main( )
{
 ch = getche();
 if(ch == 's')
     printf("Ha escrito una s.");
}
```

Una vez haya compilado el programa y lo ejecute, verá cómo cuando pulsa la tecla «S» en pantalla aparecerá el texto "Ha escrito una s," mientras que pulsando cualquier otra tecla el programa concluirá sin más.

While: esta instrucción tiene un funcionamiento interno muy similar a aquella instrucción: la propia instrucción *if* está seguida de un paréntesis, donde se indica la expresión que debe cumplirse, que, se caracterizará porque en ella aparece un operador relacional. Tras el propio paréntesis de la instrucción, nos encontramos con que aparece ya el cuerpo de la instrucción (al igual que lo poseían las instrucciones de bucle) que constará, o bien de una sola instrucción de código, o en el caso de ser más, todas encerradas entre corchetes.

Para los que conocen y han usado BASIC y están acostumbrados a usar las instrucciones condicionales de ese lenguaje, señalar que en C no se usa la palabra clave *then* para separar la condicional del propio cuerpo de la instrucción. (recordar por ejemplo *IF A=3 THEN PRINT "HOLA"*).

Con todo lo explicado hasta ahora sobre *if*, se puede decir que la única diferencia real que existe entre la instrucción *if* y *while* es el fin con el cual han sido diseñadas y los propios nombres de instrucción, ya que por lo demás, son idénticas.

Otra pregunta que quizás algunos se estén haciendo: ¿Qué pasa cuando la condición indicada en *if* no se cumple?, pues como se ve en la ejecución del programa dado de ejemplo, simplemente pasará que el cuerpo de código que incluya la instrucción después no se ejecutará, o sea, que la instrucción no ejecutará nada. En la fotografía 1 se puede ver un esquema del funcionamiento de *if* que resume todo lo explicado.

PROGRAMA DE EJEMPLO

Para mostrar el funcionamiento de la instrucción condicional *if* a fondo, a continuación se muestra un programa de ejemplo que realiza la función de contar las palabras y letras que se introducen en el ordenador al ejecutar el programa. El código del mismo es el siguiente:

```
/* Uso de IF - 2*/
main( )
{
 int contcar=0;
 int contpal=0;
 char ch;
 printf("Escriba una frase:\n");
 while( (ch= getche( ) ) != "\r")
   {
    contcar++;
    if(ch== " ") contpal++;
   }
 printf("\n Los carácteres contados son: %d",
contcar);
 printf("\n Las palabras contadas son: %d",
contpal+ 1);
}
```

Mirando el código del programa detenidamente, se puede ver que para contar las palabras que se introducen el programa, se usa simplemente el método de contar el número de espacios que se introducen hasta pulsar *Enter* (fin de frase). Esta técnica falla si se escribe más de una separación entre dos palabras, pero para

INSTRUCCIONES 'IF' ENCADENADAS

De igual forma que en el anterior capítulo se mostró la forma de poder crear bucles anidados (unos dentro de otros), con la instrucción condicional pasa lo mismo, ya que se pueden encadenar o anidar (como se prefiera llamarle) tantas como se desee.
A continuación se da un ejemplo de ello:

```
/* IF encadenados */
 main ( )
 {
  if(getche == "n")
      if(getche =="o")
        printf("Ha escrito NO");
 }
```

Como se puede ver en este ejemplo, se ha indicado como cuerpo de la primera instrucción *if* otra instrucción *if*, lo que da como resultado lo que buscábamos, anidar o encadenar dos instrucciones *if*. Este anidamiento puede, como el lector puede deducir, alargarse tanto como quiera, añadiendo tantos *if* como cuerpos de instrucción se necesiten. El encadenamiento en este ejemplo significa que la instrucción *if* más interna (la segunda para decirlo claro) forma parte del cuerpo de la instrucción *if* más externa (la primera). Dicho esto, queda claro que si la primera condición no se cumple (no se pulsa la «N» como primera tecla), la segunda ya no llegará a realizarse y por lo tanto, la instrucción *printf* no llegará a ejecutarse de ninguna forma, ya que para ello deberían cumplirse como verdaderas ambas condicionales.

realizar un ejemplo simple y para mostrar el uso de *if* esta técnica basta.
Veamos la instrucción *if* en el programa: el programa, como se ve, posee como principal novedad la línea

```
if( ch==" ") contpal++;
```

Esta instrucción lo que hace es incrementar el valor de *contpal* cada vez que se introduce un espacio en la cadena de entrada, pero dado que siempre hay una palabra más que espacios, a la hora de escribir por pantalla el número de espacios contabilizados para decir el número de palabras encontradas, tenemos que sumar 1 al número de espacios que hayamos registrado (lo cual se puede hacer dentro del propio *printf*, como se ha hecho en el ejemplo).

LA INSTRUCCION 'IF-ELSE'

La instrucción *if* por sí misma ejecutará, siempre que la condición de comprobación se cumpla, una instrucción simple o un fragmento de código entero, pero en cambio, cuando la condición es falsa, no realiza nada.

Las instrucciones condicionales son usadas en todos los lenguajes

¿Hay alguna manera de hacer que ejecute algo cuando la condición no se cumple? La respuesta es sí, hay una forma, y es usando *else* como complemento a la instrucción *if*. Antes de entrar en más detalles, se ha indicado un pequeño programa de ejemplo que hace uso de ambas instrucciones:

```
/* Test de ELSE */
main( )
{
    chat ch;
    ch = getche( );
    if(ch == "s")
        printf("\nHa pulsado S");
    else
        printf("\nNo ha pulsado S");
}
```

que hace este programa al ejecutarse. Pues muy simple: si se pulsa la tecla «S» la condición *if(ch == "s")* se dará como cierta y por lo tanto se ejecutará el cuerpo de instrucción, que en este caso es escribir por pantalla "Ha pulsado S" y luego se saltará la instrucción *else*, que sólo se ejecuta si la condición resultase falsa. En caso de que la condición no se cumpla, o sea, no se haya pulsado la tecla "s", lo que pasará es que se saltará el cuerpo de la instrucción de *if* y entonces se pasará a ejecutar el cuerpo incluido de la instrucción *else*, lo que es lo mismo que decir que se escribirá por pantalla "No ha pulsado S".

FUNCIOMAMIENTO DE LA CONDICIONAL IF

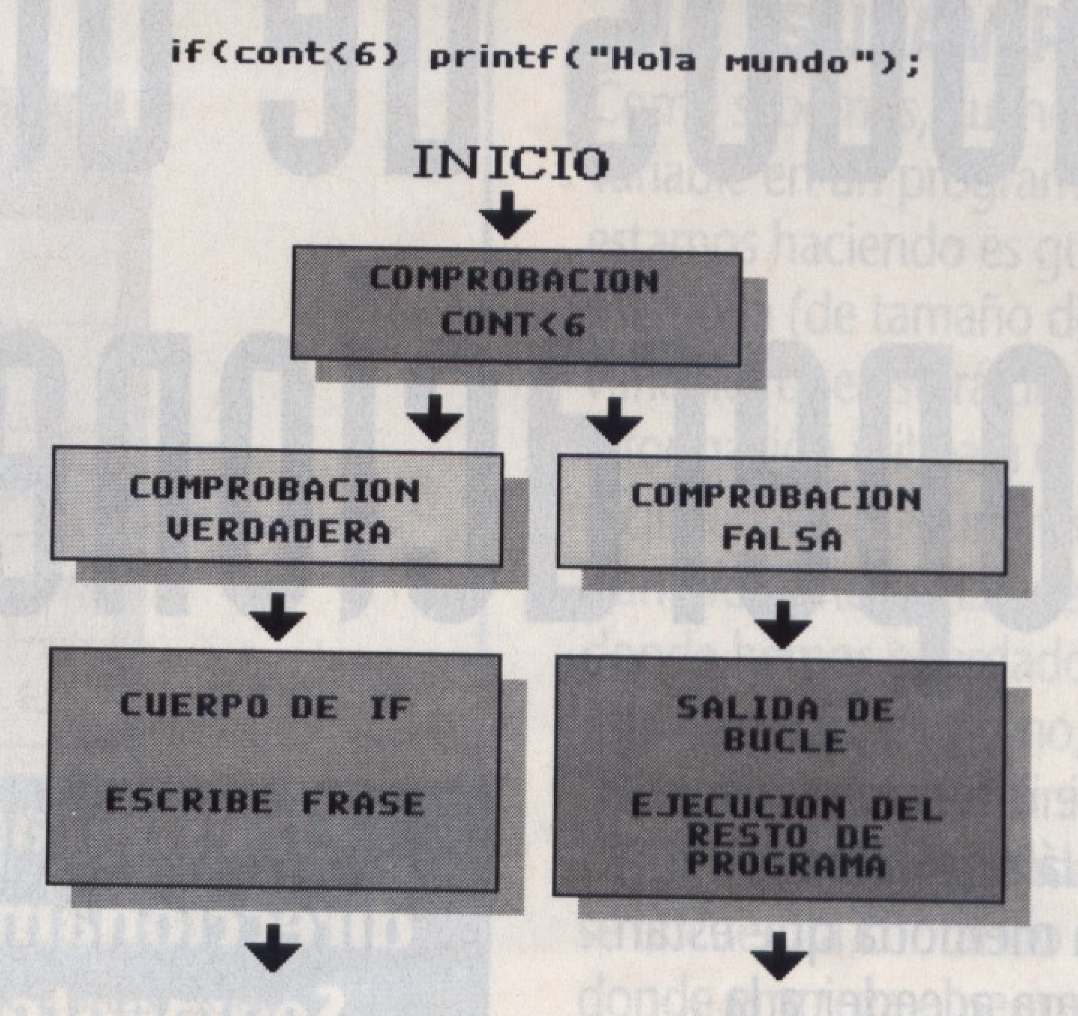


FIGURA 3.

EJEMPLOS DE COMBINACION DE 'IF-ELSE'

El siguiente programa que se va a mostrar como ejemplo, es un algoritmo más simple que el anterior, y lo único que hace es preguntar la temperatura para luego responder una cosa u otra según sea la misma. El código es el siguiente:

```
main( )
{
  int temp;
  printf("Escriba la temperatura actual:");
  scanf("%d", &temp);
  if(temp< 25)
      if(temp>15)
          printf("Hace un buen día");
      else
          printf("Vaya calor que hace");
}
```

Para entender el funcionamiento de este programa vamos a explicar cómo se ejecuta, simulando que damos algún valor de entrada. Supongamos que lo ejecutamos e introducimos el valor 20 de temperatura cuando se nos pide introducirla. ¿Qué pasará entonces?
Primero se comparará si *temp* es menor a 25 (*temp*<15), esto dará como resultado verdadero porque lo es, por lo tanto se pasará a ejecutar su cuerpo de instrucción. Una vez aquí tenemos que su cuerpo es otro *if*, el cual mirará si la temperatura es mayor a 15 grados (*temp*>15). Como esta comparación dará también verdadero, entonces se ejecuta su cuerpo (escribir "Hace un buen día") y se saltará su *else* asociado. Así de simple. Llegados aquí el programa concluye.

USANDO OPERADORES LÓGICOS

Si el anterior ejemplo de dibujado de una cruz gigante lo hubiésemos escrito aprovechando los operadores lógicos de los que disponemos en C, tenemos que el programa resultante es mucho más fácil de escribir y entender. Para reescribirlo hemos aprovechado el operador *OR*. EL código queda como sigue:

```
/* Dibuja una cruz */
main( )
{
  int x,y;
  for(y=1; y<24; y++)
   {
     for(x=1; x<24; x++)
        if(x==y || x==24 - y)
           printf("\xDB");
        else
           printf("\xB0");
     printf("\n");
   }
}
```

Como se puede observar, el uso del operador || (*OR*) hace que el código fuente necesario quede mucho más simple y más sencillo de entender. ¿Qué significa la expresión (*x==y || X== 24-y*)? Esta expresión se podría traducir como: "Si la expresión de la derecha o la de la izquierda es verdadera, la expresión global es correcta". En otras palabras: "Cuando al menos una de las dos expresiones sea válida, el *if* se dará por válido".

Los modos de direccionamiento y las operaciones aritméticas

En este capítulo, aprenderemos: los llamados modos de direccionamiento, algo fundamental, las combinaciones de Segmento y offset permitidas, y para acabar, las cuatro operaciones aritméticas básicas.

Los llamados modos de direccionamiento, el lector aprenderá cuáles son los métodos de apuntar a la memoria que están permitidos en el 8086 para acceder a la misma (escalas, bases, etc...).
Después de esto, como complemento indispensable, se van a detallar también las combinaciones de segmentos y registros que están permitidas realizar, lo cual es importante conocer para no cometer errores sintácticos graves a la hora de escribir punteros de memoria en nuestros programas.
Para concluir, después de dos temas pesados y difíciles de entender para el lector, se va a pasar a explicar algo más ameno y sencillo de aprender: la forma de aplicar las cuatro operaciones aritméticas en este lenguaje, o lo que es lo mismo, se enseñará cómo funcionan las instrucciones ADD, SUB, MUL y DIV.

MODOS DE DIRECCIONAMIENTO

Los parámetros (operadores) de una instrucción pueden ser de tres tipos: registros internos de la CPU (detallados en un capítulo anterior), valores inmediatos y direcciones de memoria (los conocidos como punteros) de las que hay cinco tipos diferentes de representación. En total son siete clases de parámetros los que podemos usar, y conjuntamente forman los llamados siete *modos de direccionamiento*. En la fotografía número uno, podemos ver un esquema de todos ellos, los cuales se detallan a continuación:

MODOS DE DIRECCIONAMIENTO

Nº	MODO	OPERADOR	REG S	EJEMPLOS
1	Registro	Registro	--	MOV AX,BX / ADD BL,CL
2	Valor	Valor inmediato	--	MOV AX,133 / ADD AL,20
3.1	Variable	Offset inmediato (variable)	DS:	MOV AX,VARIABLE
3.2	Indirecto mediante registro	[BX]	DS:	MOV DX,DS:[BX]
		[BP]	SS:	MOV CX,SS:[BP]
		[DI]	DS:	ADD AX,DS:[DI]
		[SI]	DS:	ADD BX,DS:[SI]
3.3	Relativo a base	[BX] + desp	DS:	MOV CX,DS:[BX+4000]
		[BP] + desp	SS:	MOV AX,SS:[BP+3030]
3.4	Directo indexado	[DI] + desp	DS:	SUB AX,DS:[DI+20000]
		[SI] + desp	DS:	MOV DX,DS:[SI+100]
3.5	Indexado a base	[BX] + [SI] + desp	DS:	MOV AX,DS:[BX][SI]+300
		[BX] + [DI] + desp	DS:	ADD AX,DS:LISTA[BX][DI]
		[BP] + [SI] + desp	SS:	MOV AX,SS:[BP][SI]+300
		[BP] + [DI] + desp	SS:	MOV AX,SS:TABLA[BP][DI]

ESQUEMA CON TODOS LOS MODOS DE DIRECCIONAMIENTO QUE PODEMOS USAR EN UN PROCESADOR 8086.

1-Registros internos: los registros internos ya fueron detallados en número y utilidad en el artículo anterior. Ejemplos de instrucciones que poseen solamente registros internos como operadores: MOV AX,AX / MOV BL,BL / ADD CL,DL / SUB CX,DX , etc...
2-Valores inmediatos: usar un valor inmediato es indicar el número que queremos dar como operador escrito directamente en la propia instrucción.
Como regla muy importante que podemos añadir a las indicadas en el artículo anterior sobre reglas sintácticas de instrucciones, es que nunca podemos colocar un valor inmediato como primer operador, ya que entonces el resultado de la instrucción, que como debemos recordar se almacena siempre en el primer operador, no podría almacenarse ni en un registro ni en la memoria , ya que no tendría lógica funcional.
Otra regla que podemos añadir muy importante es que nunca se pueden colocar en una instruccion dos operadores de diferente tipo de longitud, como sería sumar, por ejemplo, los registros AX y BL (registros/variables de 16 y 8 bits y viceversa). Ejemplos de instrucciones que usan valores inmediatos como operadores son: MOV AX,500 / ADD BX,100 / SUB AL,10, etcétera.
3-Direcciones de memoria: como se ha dicho, se puede usar una dirección de memoria como parámetro de instrucción. En total hay cinco modos de indicar una dirección de memoria al procesador. A continuación se detallan todas:
3.1-Modo directo: una dirección directa es indicar la dirección de memoria a la que queremos acceder (*OFFSET*) directamente como un valor inmediato como

COMBINACIONES DE REGISTROS

	CS	SS	DS	ES
IP	SI	NO	NO	NO
SP	NO	SI	NO	NO
BP	POSIBLE	DEFECTO	POSIBLE	POSIBLE
BX	POSIBLE	POSIBLE	DEFECTO	POSIBLE
SI	POSIBLE	POSIBLE	DEFECTO	POSIBLE
DI	POSIBLE	POSIBLE	DEFECTO	DEFECTO

ESQUEMA DE TODAS LAS COMBINACIONES DE REGISTROS Y SEGMENTOS CON INDICACIONES DE CUALES SON PERMITIDAS O NO (VALIDAS O NO).

complemento al registro de segmento. Ejemplos de ello son: MOV AX,DS:[50] / ADD BX,DS:[12050], etcétera.

3.2-Modo indirecto mediante registro: éste es igual al anterior modo, pero en lugar de usar la combinación segmento *:offset* con un valor inmediato para el último componente del puntero, éste se indica mediante un registro interno, que puede ser BX, BP, DI o SI. Ejemplos de instrucciones que usan este método de direccionamiento son: MOV AX,DS:[BX] / MOV AX,DS:[BP] / ADD CX,ES:[SI], etcétera.

3.3-Modo relativo a base: podríamos decir que este modo es como unir los dos modos anteriores en uno. Ahora, para indicar el desplazamiento (*OFFSET*) se realiza mediante un registro interno más un valor inmediato que se le suma. En este modo, podemos usar para el *OFFSET* los registros BX y BP. Ejemplos de instrucciones son: MOV AX,DS:[BX+100] / ADD DX,DS:[BX+100] / SUB AX,SS:[BP+32], etcétera.

3.4-Modo directo indexado: este modo es igual al anterior, pero con la diferencia de que se usan los registros base DI y SI , que como sabemos, se llaman registros índice. Ejemplos de esto son: MOV AL,DS:[DI+30] / MOV BX,TABLA:[DI].

3.5-Modo indexado a base: éste es el modo más complejo, y es como el resultado de unir los cuatro anteriores. En este modo, el *OFFSET* se compone a partir de un registro que hace de base, un registro índice y un valor inmediato que se les suma. Las combinaciones posibles se indican a continuación con ejemplos: MOV AX,DS:[BX][SI]+30 / MOV AX,DS:[BX][SI]+1000 / MOV AX,SS:[BP][SI]+32 / MOV AX,SS:[BP][DI]+512.

COMBINACIONES DE REGISTROS VÁLIDAS

Una vez conocemos los cinco modos de direccionamiento que hay para crear punteros a memoria, debemos saber cuáles son las combinaciones permitidas que existen entre registros de segmento y registros que usamos como desplazamiento (*OFFSET*).

En la fotografía número 2 está el esquema con las combinaciones posibles y su explicación de resultado. A continuación explicamos su contenido.

Un mismo puntero puede representarse de muchas formas válidas

En las combinaciones que pone *<Sí>*, quiere decir que sólo está permitido unir el registro de segmento y el de desplazamiento correspondientes a dicho cuadrante. Si pone *<No>*, significa que no podemos crear dicha combinación bajo ninguna condición.

Por otro lado, cuando está escrito *<defecto>*, significa que si al usarlo en una instrucción omitimos escribir el registro de segmento, el ensamblador supondrá que debe colocar el que corresponde a dicho *OFFSET* por defecto, que será el que le corresponda en la tabla de la fotografía. Así, por ejemplo, al ser DS el registro de segmento por defecto de BX , SI y DI , si escribimos la línea MOV AX,[BX] , será lo mismo que si pusiéramos MOV AX,DS:[BX], y en caso de que fuese MOV AX,[SI], lo substituiría automáticamente por MOV AX,DS:[SI].

Por último, cuando encontramos en la fotografía una combinación *segmento:registro* en la que se indica *<Posible>*, quiere decir que dicha combinación está permitida, pero que debe escribirse entonces la secuencia completa indicando el segmento que queremos usar y el *offset* en la instrucción.

¿QUE ES UNA VARIABLE?

Como sabemos, cuando declaramos una variable en un programa, lo que realmente estamos haciendo es guardar una zona de memoria (de tamaño dependiente del tipo de variable) que estará destinada a contener información útil para el programa. Dicho esto, es fácil deducir que una variable no es más que un puntero a memoria que referencia al primer bit donde hemos guardado su contenido, y que está formado, cómo no, por un segmento base (bloque donde se definió la variable) y por un *OFFSET* que corresponde al número de bit donde se encuentra contando como base cero al bit donde empieza el segmento, y así, en el ejemplo de querer sumar 5 a una variable llamada COCHE, que sería *ADD COCHE,5* , lo que estamos haciendo realmente es sumar 5 al contenido de una dirección de memoria que contiene el valor que hemos asignado a COCHE , lo cual quedaría realmente por poner un ejemplo: *ADD word ptr DS:[5000],5*. El hecho de dar un nombre a la zona de memoria asignada para contener una información, se hace para que sea más fácil programar y para ahorrarnos tener que indicar cada vez a qué tipo de variable apunta dicho puntero. Así, por ejemplo, las del primer fuente VARIABLE1, VARIABLE2 y RESULTADO, no son más que tres direcciones de memoria (*OFFSETS*) relativas a una base (en el ejemplo el segmento DATOS) y la línea *MOV AX,VARIABLE1* realmente no es más que un *MOV AX,DS:[00]*, la cual, como podemos comprobar, no es más que una instrucción que usa el modo de direccionamiento 3.1 explicado anteriormente. De la conversión de nombres (ETIQUETAS) a direcciones de memoria, se encarga el ensamblador, cuando convierte nuestros fuentes a extensión OBJ, antes de que se conviertan a EXE con el LINK.EXE.

Bibliografía relacionada

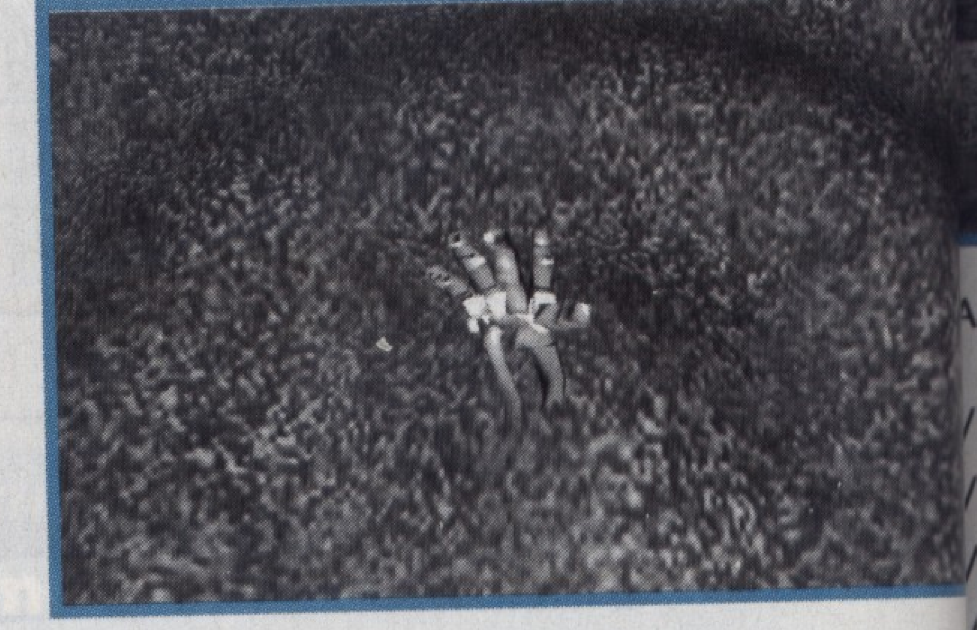
- ***8088-8086/8087 Programación ENSAMBLADOR en entorno MS-DOS***, **Miguel Ángel Rodríguez Roselló. Anaya Multimedia.**
- **Programación del 80386/387.** ***Manual de referencia y técnicas avanzadas de programación para diseñadores de sistemas, programadores y usuarios avanzados.*** **John H. Crawford/Patrick P. Gelsinger. Anaya Multimedia.**
- ***Programación del Z80.*** **Rodnay Zaks. Anaya Multimedia.**
- ***80286 Arquitectura y sistemas.*** **Grupo Waite; Edmund Strauss. Anaya Multimedia.**
- ***Nueva Guía del programador en Ensamblador para IBM PC , XT , AT y compatibles.*** **Peter Norton; John Socha. Anaya Multimedia.**

JUEGOS GANADORES: 1º

Autores: V. Reyes, C. López, M. Ló

Mr. Bones [o vete al Infierno]

Este programa es el que ha resultado ganador en este número de DIVMANÍA. No ha habido ninguna discusión a la hora de otorgarle el primer puesto y el merecido premio que conlleva. Nuestra más cordial enhorabuena a los ganadores. Esperamos recibir pronto vuestro próximo trabajo que, estamos seguros, será todavía mejor.

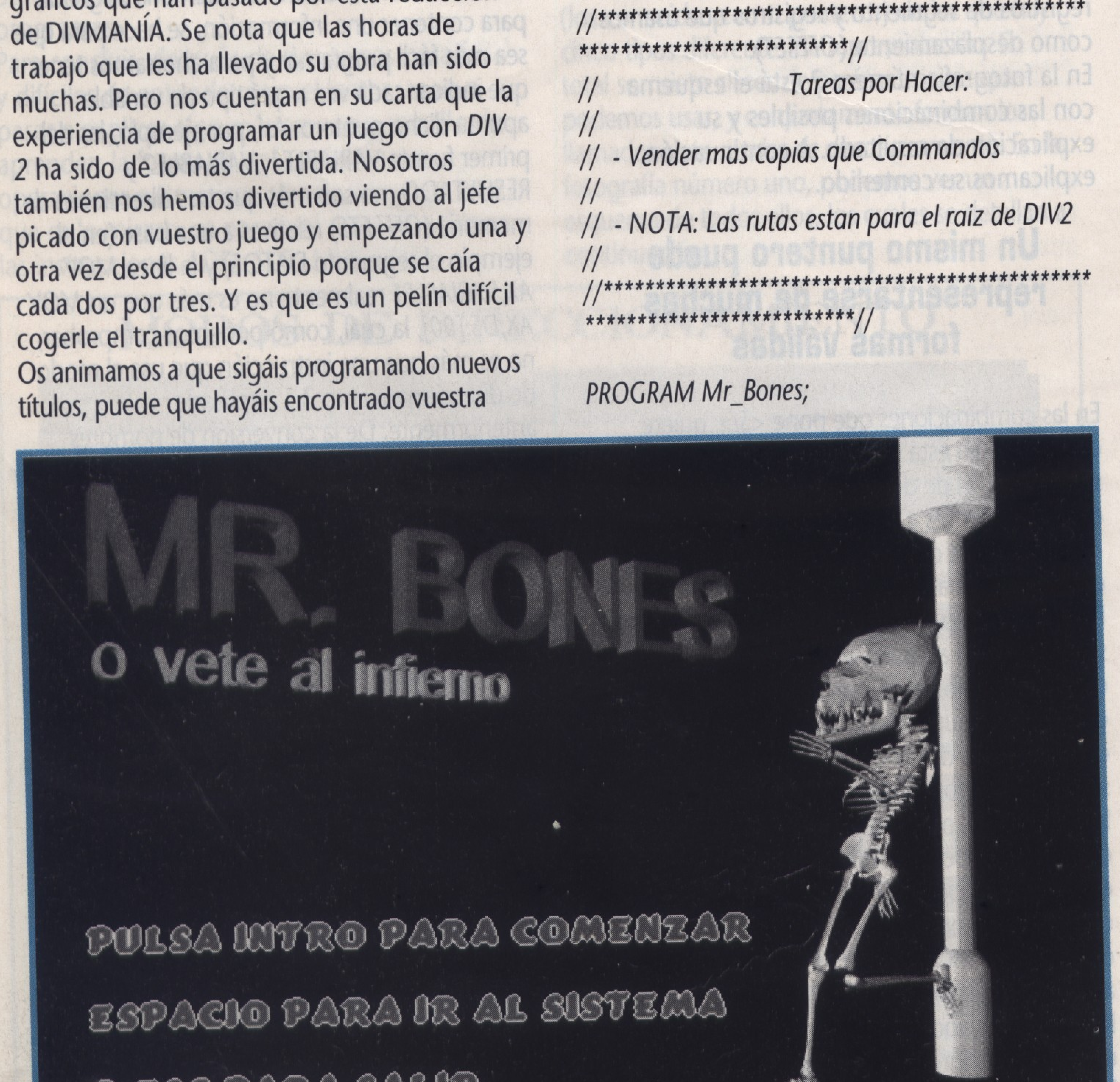


Este juego ha sido diseñado, desarrollado, producido, corregido, editado, planchado, y no sabemos si será distribuido y comercializado, por Victor Reyes Céspedes, y los hermanos Carlos y Miguel López. Calidad tiene, si lo retocan un poco más.

Ha sido uno de los juegos con mejores gráficos que han pasado por esta redacción de DIVMANÍA. Se nota que las horas de trabajo que les ha llevado su obra han sido muchas. Pero nos cuentan en su carta que la experiencia de programar un juego con *DIV 2* ha sido de lo más divertida. Nosotros también nos hemos divertido viendo al jefe picado con vuestro juego y empezando una y otra vez desde el principio porque se caía cada dos por tres. Y es que es un pelín difícil cogerle el tranquillo.

Os animamos a que sigáis programando nuevos títulos, puede que hayáis encontrado vuestra vocación en ésto de desarrollar videojuegos. Por lo que hemos visto tenéis madera.

Al resto de los lectores les incluimos un trozo del larguísimo código fuente. Lo encontraréis integro en el CD-Rom que acompaña a la revista.



Código fuente

```
//*************************************************
*************************//
//              Tareas por Hacer:
//
//  - Vender mas copias que Commandos
//
//  - NOTA: Las rutas estan para el raiz de DIV2
//
//*************************************************
*************************//

PROGRAM Mr_Bones;

GLOBAL
  OBSTACULO; // Variable que verifica si
colisionas con algo (suelo, muro...)
  FPG_PROTA;
  Fondo;   // Variable para el n£mero de
pantalia activo.
  Dureza;  // Variable para el numero de m
de dureza activo.
  // Variables para las animaciones.
  Fli1; Fli2; Fli3; Fli4; Fli5; Fli6;
  // (Intro, Muerte1, Muerte2, MuerteFinal,
LogoTEA, Muerte3)

  Memoria_Libre;
  Letras; Letras2;  // Variable para el Load_
  Vidas;  // Variable para las vidas restante
Bones
  Puntos;  // Puntuacion.
  Sonido1; Sonido2; // Variable para la mus
de Blade que se carga en Logo_TEA()
  Musica_Fase;  // Musica durante el juego

  Var_Sistema;

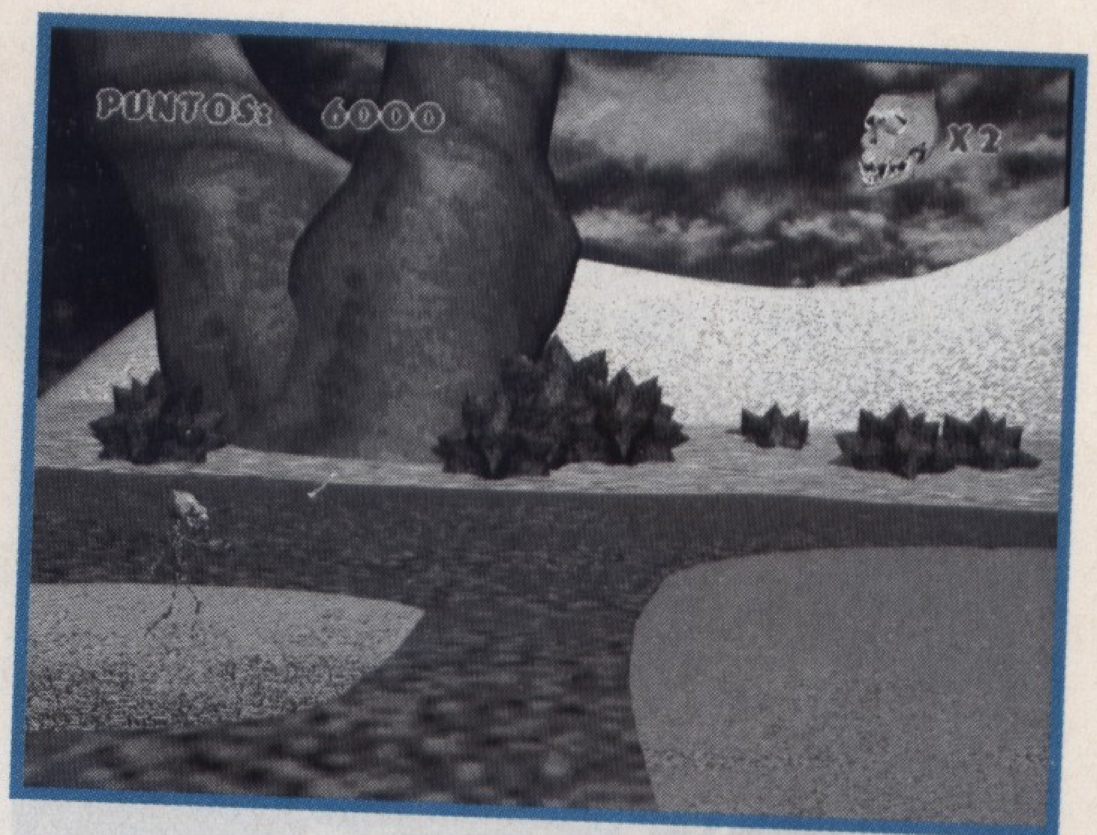
  // Variables que indican el primer mapa y
dureza de la primera pantalla
  // de cada uevo mundo
  Mapa_FAse1=100;
  Dureza_Fase1=101;

  // Coordenada de Bones, de uso global
  x_bones;
  FLAG_Bones;
  // Variable que controla la creaci¢n enem
  Crea_Ahora;
  // Variable que controla la destruccion del
disparo
  VAR_Disparo;

//*************************************************
*************************//
//
//
```







La mecánica del juego consiste en avanzar dando los saltos necesarios y esquivar a los esqueletos enemigos.

```
//
//                Proceso Principal
//
//
//
//******************************************************
**************************//


BEGIN

  SET_MODE(m640x480);

  // Funciones de carga de Ficheros
  FPG_PROTA=LOAD_FPG("BONES.FPG");
  LETRAS=LOAD_FNT("BONES.FNT");
  LETRAS2=LOAD_FNT("BONES2.FNT");

  // Variables Iniciales del juego
  Vidas=3;
  Puntos=0;
  Fondo=Mapa_Fase1;
  Dureza=Dureza_Fase1;

  Var_Sistema=0; // 0 -> No estamos en el
menu Sistema, 1 en el caso de que sí estemos.

  // Cargamos la intro del logo
  Logo();

END


//******************************************************
**************************//
//
//
//      Proceso de la pantalla de Inicio
//
//
//
//******************************************************
**************************//
PROCESS Menu_Inicio();

BEGIN

    Let_Me_Alone();
    FADE_OFF();
    Clear_Screen();
    Delete_Text(All_Text);
    FADE_ON();

    PUT_SCREEN(FPG_PROTA,900);

    WRITE(Letras,30,300,0,"Pulsa Intro para
comenzar");
    WRITE(Letras,30,350,0,"Espacio para ir al
sistema");
    WRITE(Letras,30,400,0,"o ESC para salir.");

    // Reiniciamos las variables para la proxima
vez que juguemos.
    Vidas=3;
    Puntos=0;
    Fondo=Mapa_Fase1;
    Dureza=Dureza_Fase1;

    Var_Sistema=0;
  LOOP

    Memoria_Libre=Memory_Free();

    IF (KEY(_ENTER)) // Si pulsamos Intro,
empezamos.
        Delete_Text(All_Text);
        Let_Me_Alone();
        UNLOAD_WAV(Sonido1);
        Fade_OFF();
        RETURN(BONES());
    END

    IF (KEY(_SPACE))
        Var_Sistema=1;
        RETURN(Sistema());
    END

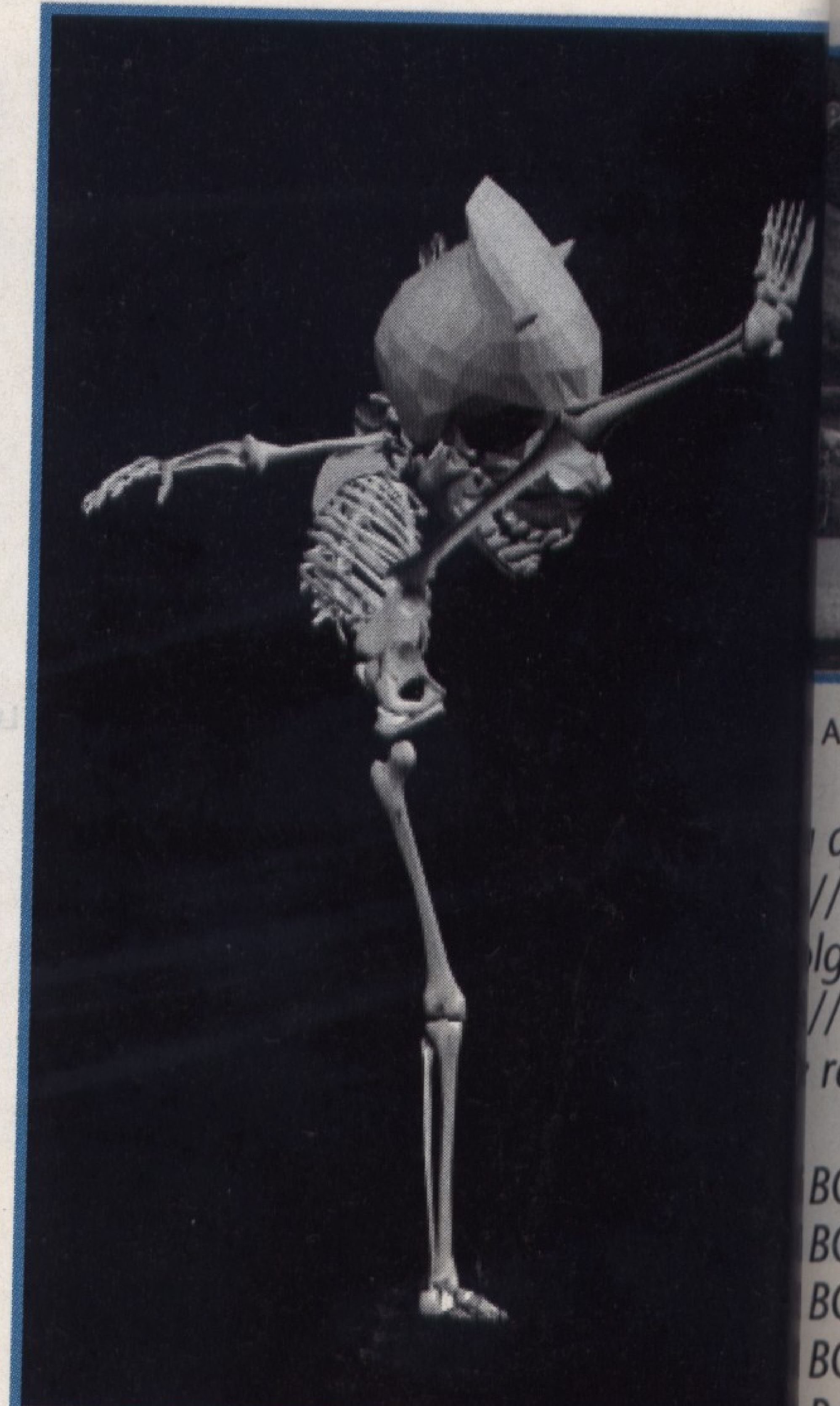
    IF (KEY(_ESC))
```



Mr. Bones ardiendo en el infierno de los perdedores.

Juegos ganadores: 1º





HAN INTRODUCIDO ESCENAS ANIMADAS, COMO ESTE BAILE DEL ESQUELETO, EL PROTAGONISTA DEL JUEGO.

```
      Let_Me_Alone();
      Delete_Text(All_Text);
      Clear_Screen();

      REPEAT
         PUT_SCREEN(FPG_PROTA,99); FRAME;
      UNTIL (SCAN_CODE<>0);
         EXIT("(C) Bokeron SoftWare 1999. Mr.
Bones Versi¢n Beta 1.3",0);
   END

   FRAME;
 END

END

//***********************************************
*************************//
//
//
//          Proceso del Men£ del Sistema
//
//
//
//***********************************************
*************************//

PROCESS Sistema();

BEGIN

  Let_Me_Alone();
  FADE_OFF();
  Clear_Screen();
```

```
  Delete_Text(All_Text);
  FADE_ON();

  WRITE(Letras,15,40,0,"boKeron SoftWare
1999. Mr Bones Beta 1.3");

  WRITE(Letras,30,200,0,"Memoria Libre:");
  WRITE(Letras,30,250,0,"Frames Por
Segundo:");
  WRITE(Letras,70,450,0,"Pulsa ESC para
regresar al men£...");

  WRITE(Letras,360,150,0,"Actual");
  WRITE(Letras,495,200,0,"(+32000)");
  WRITE(Letras,495,250,0,"(+24)");

  WRITE_INT(Letras,360,200,0,OFFSET
Memoria_Libre);
  WRITE_INT(Letras,360,250,0,OFFSET FPS);

  IF (Memoria_Libre>=32000)
    WRITE(Letras,450,200,0,"OK");
    ELSE
    WRITE(Letras,450,200,0,"!!!");
  END

  IF (FPS>=24)
    WRITE(Letras,450,250,0,"OK");
    ELSE
    WRITE(Letras,450,250,0,"!!!");
  END

  LOOP
```

```
    Memoria_Libre=Memory_Free();

    IF ((KEY(_ESC)) AND Var_Sistema)
      Return(Menu_Inicio());
    END

    FRAME;
  END

END

//**********************************************
************************//
//
//
//                Proceso de Bones
//
//
//
//**********************************************
************************//

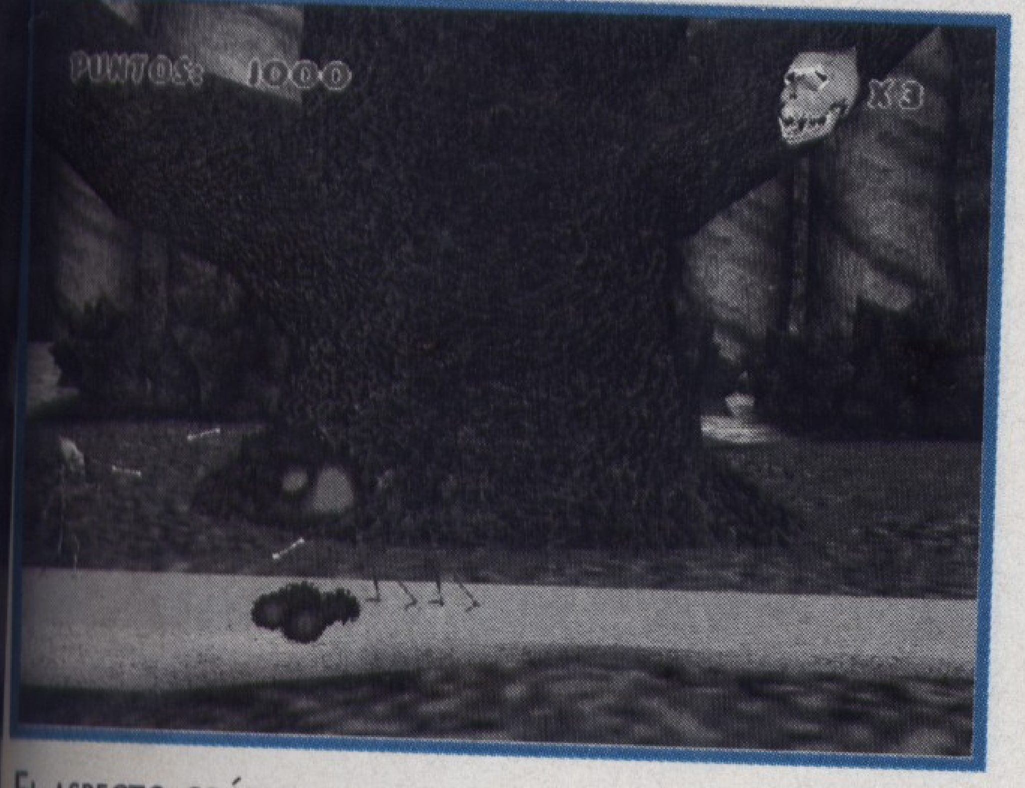
PROCESS Bones();

PRIVATE

  En_Suelo; Salto; Inc_salto; Velocidad;
Ultima_Velocidad; IncX; MAX_SALTO;

  // Si esta a 0I, creamos enemigos en el fondo
actual, si vale 1, no.
  // Esto hace que no creemos infinitos enemig
```

Juegos ganadores: 1º



El aspecto gráfico es escelente.

```
ya que de lo contrario
    // el juego se relentizarįa hasta acabar
colgandose.
    // De todas formas, cuando se crean enemigos,
se relentiza un poco.

    BONES_ANDA_INICIO=2;
    BONES_ANDA_FINAL=16;
    BONES_PARADO=1;
    BONES_SALTANDO=20;
    Puedo_Disparar;

BEGIN

    PLAY_CD(2,1);

    FADE_ON();
    x=50; y=385; Graph=1;
    Crea_Ahora=0;
    Puedo_Disparar=timer;

    Cabeza_Bones(); // Proceso de la cabeza
indicadora de vidas

/*  WRITE(0,0,5,0,"Memoria Libre:");
    WRITE(0,0,15,0,"Frames Por Segundo:");

    WRITE_INT(0,95,5,0,OFFSET Memoria_Libre);
    WRITE_INT(0,120,15,0,OFFSET FPS);
*/

    LOOP
      x_bones=x;

      // Controla la creacion de enemigos, segun
sea la pantalla actual
      SWITCH(Fondo)
        CASE 102:
          IF (Crea_Ahora==0);
            Enemigo1(300,385,201,100);
            Enemigo1(320,385,201,110);
            Enemigo1(440,385,201,80);
            Enemigo1(400,385,201,85);

            Crea_Ahora=1;
          END
          IF ((x>200) AND (Crea_Ahora==1))
            Enemigo1(0,385,201,120);
            Enemigo1(639,385,201,90);
            Crea_Ahora=2;
          END

        END
        CASE 104:
          IF (Crea_Ahora==0);
            Enemigo1(640,385,201,101);

            Crea_Ahora=1;
          END
        END
        CASE 106:
          IF (Crea_Ahora==0);
            Enemigo1(350,385,201,110);
            Enemigo1(400,385,201,70);

            Crea_Ahora=1;
          END
          IF ((x>100) AND (Crea_Ahora==1))
            Enemigo1(0,385,201,120);
            Enemigo1(639,385,201,90);
            Crea_Ahora=2;
          END
          IF ((x>300) AND (Crea_Ahora==2))
            Enemigo1(0,385,201,100);
            Enemigo1(640,385,201,100);
            Crea_Ahora=3;
          END

        END
        CASE 108:
          IF (Crea_Ahora==0);
            Enemigo1(500,385,201,110);

            Crea_Ahora=1;
          END
        END
        CASE 110:
          IF (Crea_Ahora==0);
            Enemigo1(260,385,201,90);

            Crea_Ahora=1;
          END
        END
        CASE 112:
          IF (Crea_Ahora==0);
            Enemigo1(165,385,201,110);

            Crea_Ahora=1;
          END
          IF ((x>200) AND (Crea_Ahora==1))
            Enemigo1(639,385,201,80);
            Crea_Ahora=2;
          END

        END
        CASE 114:
          IF (Crea_Ahora==0);
            Enemigo1(300,385,201,110);
            Enemigo1(440,385,201,100);
            Enemigo1(400,385,201,115);

            Crea_Ahora=1;
          END
          IF ((x>100) AND (Crea_Ahora==1))
            Enemigo1(0,385,201,120);
            Enemigo1(639,385,201,80);
            Crea_Ahora=2;
          END
          IF ((x>200) AND (Crea_Ahora==2))
            Enemigo1(0,385,201,80);
            Enemigo1(639,385,201,120);
            Crea_Ahora=3;
          END
          IF ((x>300) AND (Crea_Ahora==3))
            Enemigo1(0,385,201,75);
            Enemigo1(639,385,201,70);
            Crea_Ahora=4;
          END

        END
        CASE 116:
          IF (Crea_Ahora==0);
            Enemigo1(310,385,201,110);

// Y todo esto, pa la bazofia que es el juego...
```



Autor: Jorge Bergos

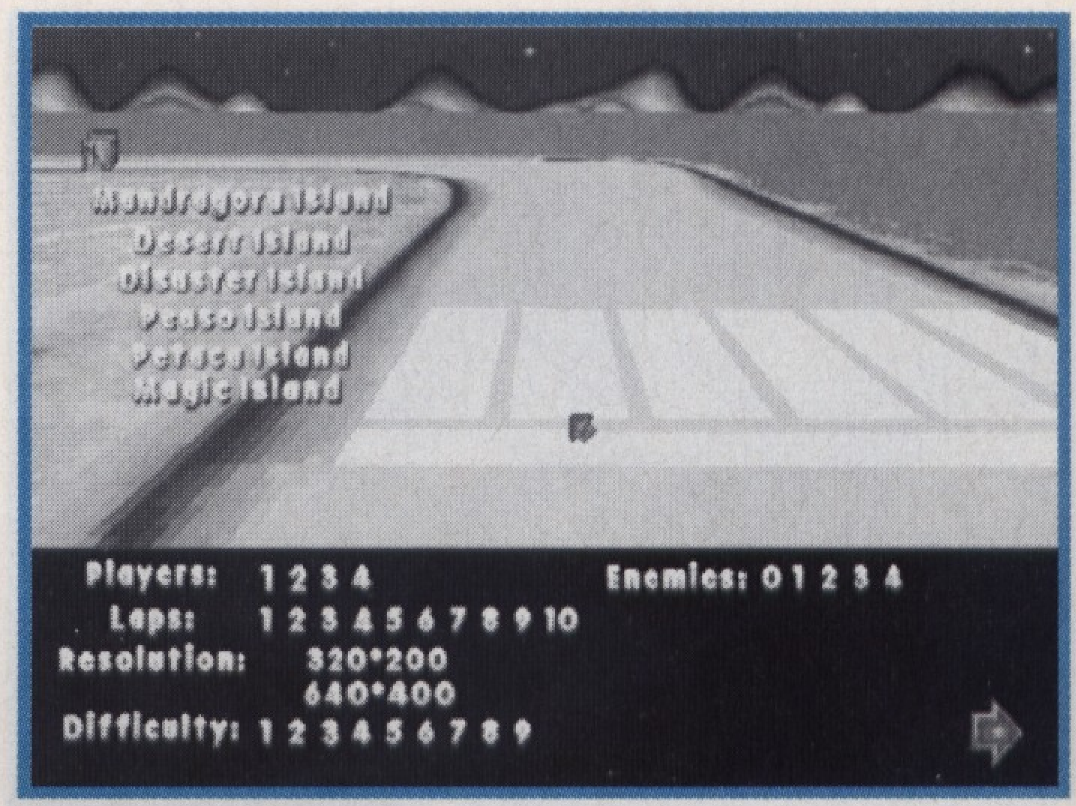
Acelerator

El ganador del segundo premio de este mes es un adictivo programa de carreras de coches en plan Mario Kart. Se le han incluido algunos trucos de magia y la más que interesante opción de jugar hasta con 4 contrincantes a la vez.

A continuación os dejamos que el propio Jorge os explique un poco el proceso de creación del programa.
Empecé pensando desarrollar un juego divertido, con el cual pudieran jugar a la vez cuantas más personas mejor y de paso poder también divertirme yo al crearlo. Y así empezó todo, primero hice unos esbozos sobre papel, luego empecé a reunir gráficos y más tarde a escribir funciones que necesitaría.

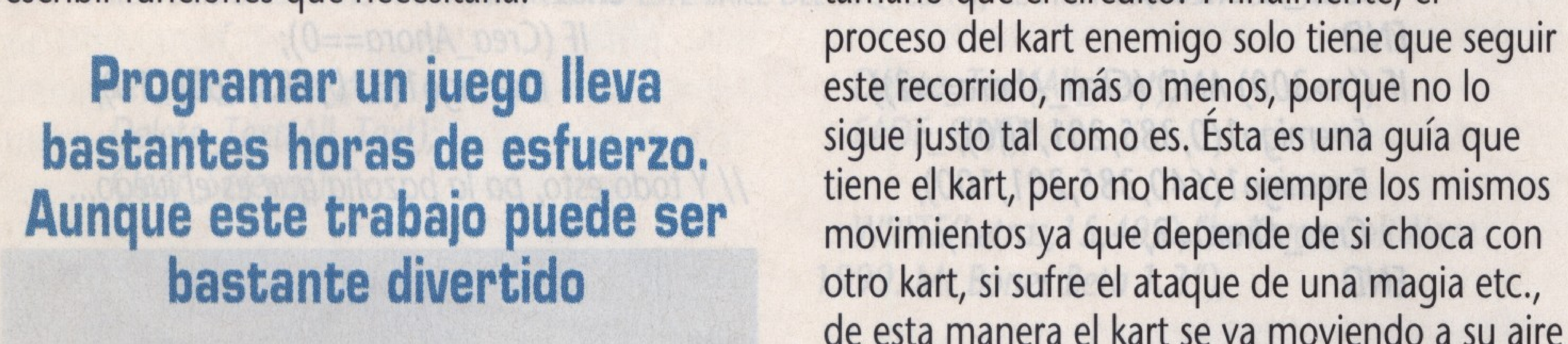
Programar un juego lleva bastantes horas de esfuerzo. Aunque este trabajo puede ser bastante divertido

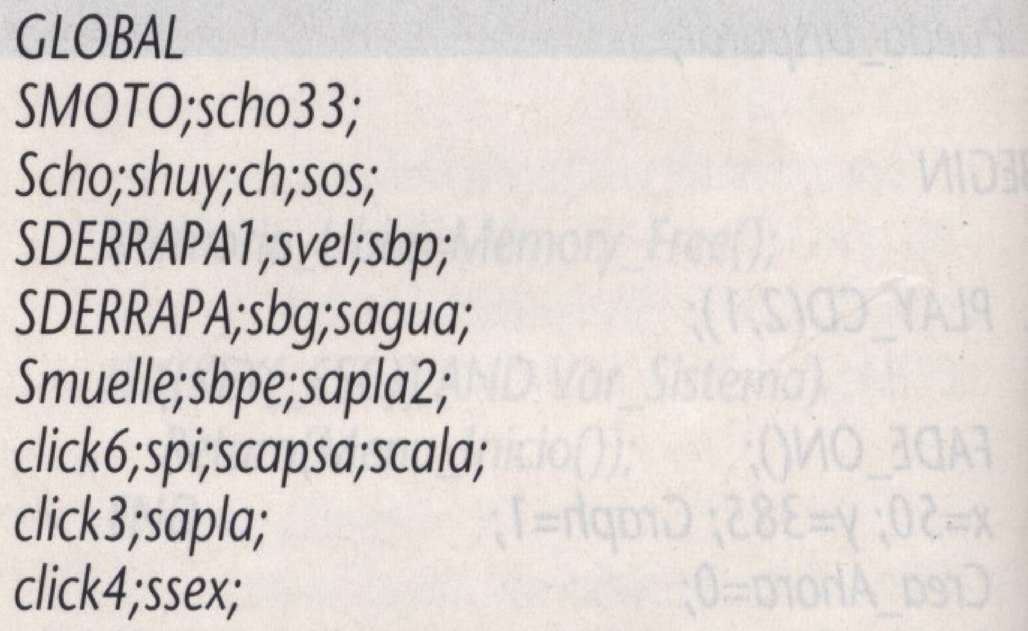
La parte más difícil de comprender del programa, y que para mí fue la más interesante y entretenida de realizar, es la creación de un sistema para que los enemigos se muevan por el circuito. Como lo hice: primero, encima del dibujo del circuito se traza con un color determinado por donde deben pasar los karts. Segundo, hacer varios recorridos más, de colores diferentes, y así conseguir que los karts se puedan mover en un mismo circuito por diversos recorridos. Tercero, todos los recorridos trazados sobre el dibujo se mueven en un mapa aparte de igual tamaño que el circuito. Y finalmente, el proceso del kart enemigo solo tiene que seguir este recorrido, más o menos, porque no lo sigue justo tal como es. Ésta es una guía que tiene el kart, pero no hace siempre los mismos movimientos ya que depende de si choca con otro kart, si sufre el ataque de una magia etc., de esta manera el kart se va moviendo a su aire por el circuito pero siguiendo el camino correcto sin salir del circuito.
Como conclusión me gustaría matizar que gracias a Div he podido trabajar la parte



creativa de la realización de un juego sin tener que preocuparme tanto de aspectos tediosos, como crear librerías gráficas, ya que Div siempre me ha ofrecido unas funciones muy eficientes y completas.
El código fuente ocupaba 107 páginas, así que por motivos obvios no hemos podido incluir más que el principio del programa.

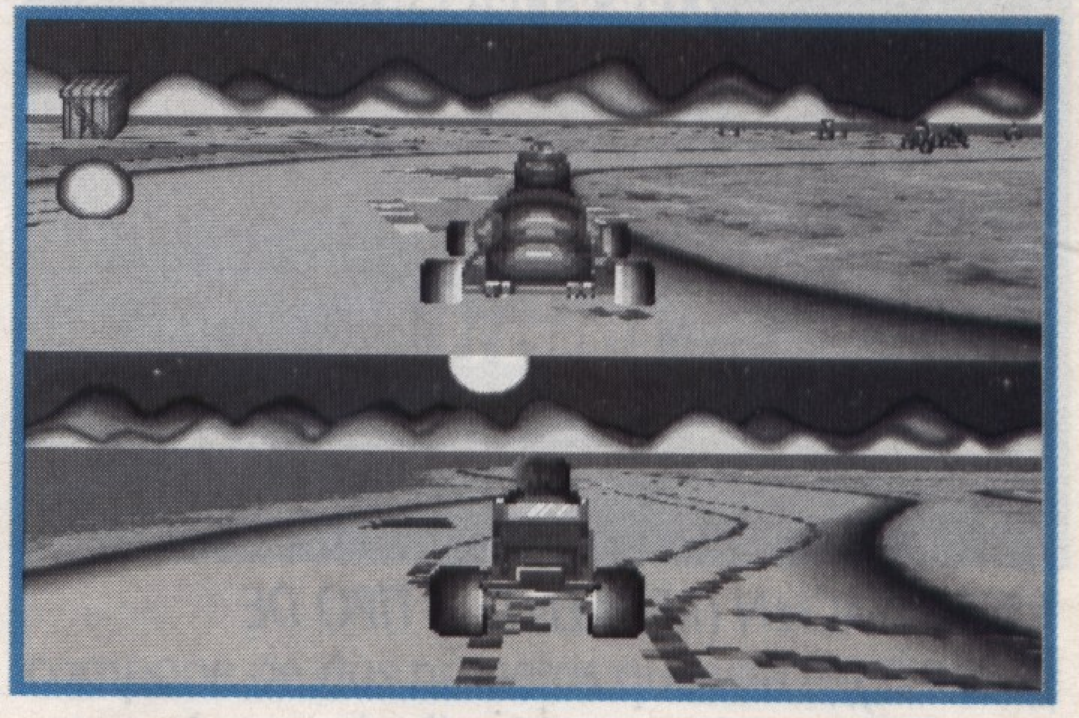
```
PROGRAM ACCELERATOR;
GLOBAL
SMOTO;scho33;
Scho;shuy;ch;sos;
SDERRAPA1;svel;sbp;
SDERRAPA;sbg;sagua;
Smuelle;sbpe;sapla2;
click6;spi;scapsa;scala;
click3;sapla;
click4;ssex;
click5;strans;slik;
A;B;so;
cmap2;menu;joo;
color;C;jordi;
xx;yy;gua;
TI;vol;
reso;terra;
x1;y1;
salir;
x2;y2;
x3;y3;
xoca;cc88;
level;
vmaster;vmaster2;
mx1;mx2;mx3;
BPLAYER;
```











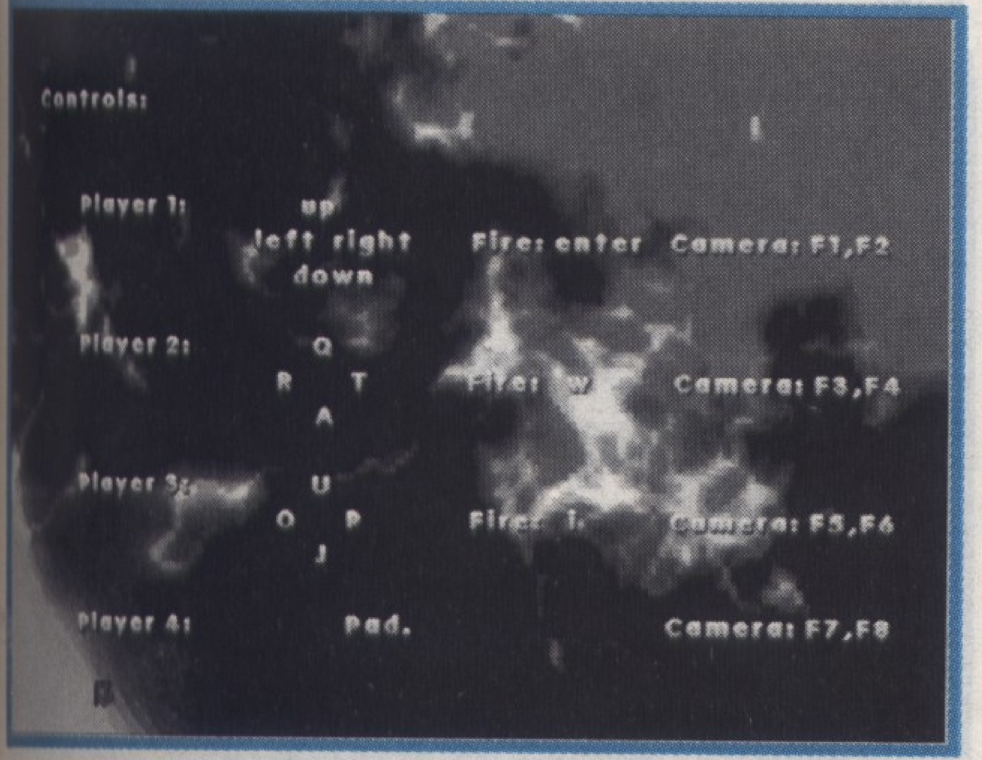
EL JUEGO DISPONE DE SEIS CIRCUITOS COMPLETOS DONDE PRACTICAR ESTE DIVERTIDA MODALIDAD DE CONDUCCION.

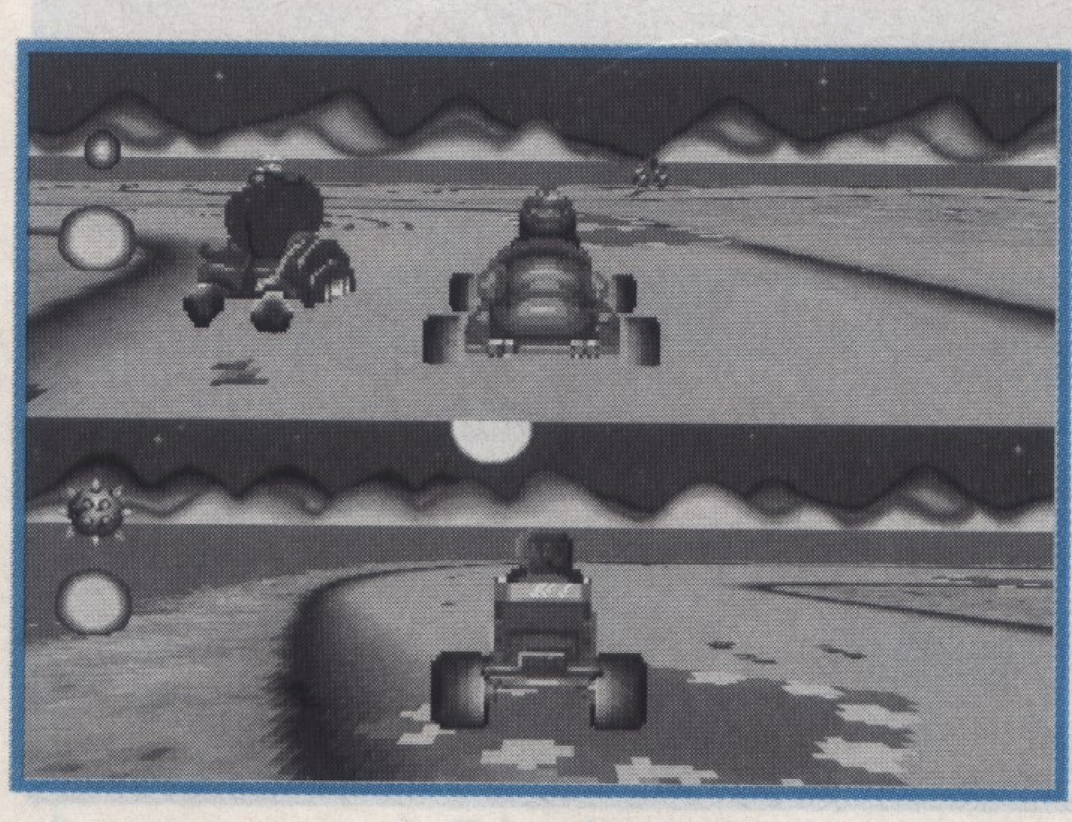
```
reg;cx1;
cx2;my1;
my2;BTIME;
BDES;cy1;
cy2;hori;
EM;SEM;
cpus;nvoltes;
grausxocar;
// DIFERENTES VISTAS DE CADA PERSONAJE //
vpeaso1[]=8, 8, 9, 2, 3, 4, 5, 6, 7;
vpeaso2[]=8, 15, 16, 1, 10, 11, 12, 13, 14;
vpeaso3[]=8,-54, 58, 53, -58, 54, 56, 50,-56;
vpeaso4[]=8,-55, 59, 52, -59, 55, 57, 51, -57;
vmono1[]=8,208,209,202,203,204,205,206,2
07;
vmono2[]=8,215,216,201,210,211,212,213,
214;
vtaja1[]=8,308,309,302,303,304,305,306,307;
vtaja2[]=8,315,316,301,310,311,312,313,314;

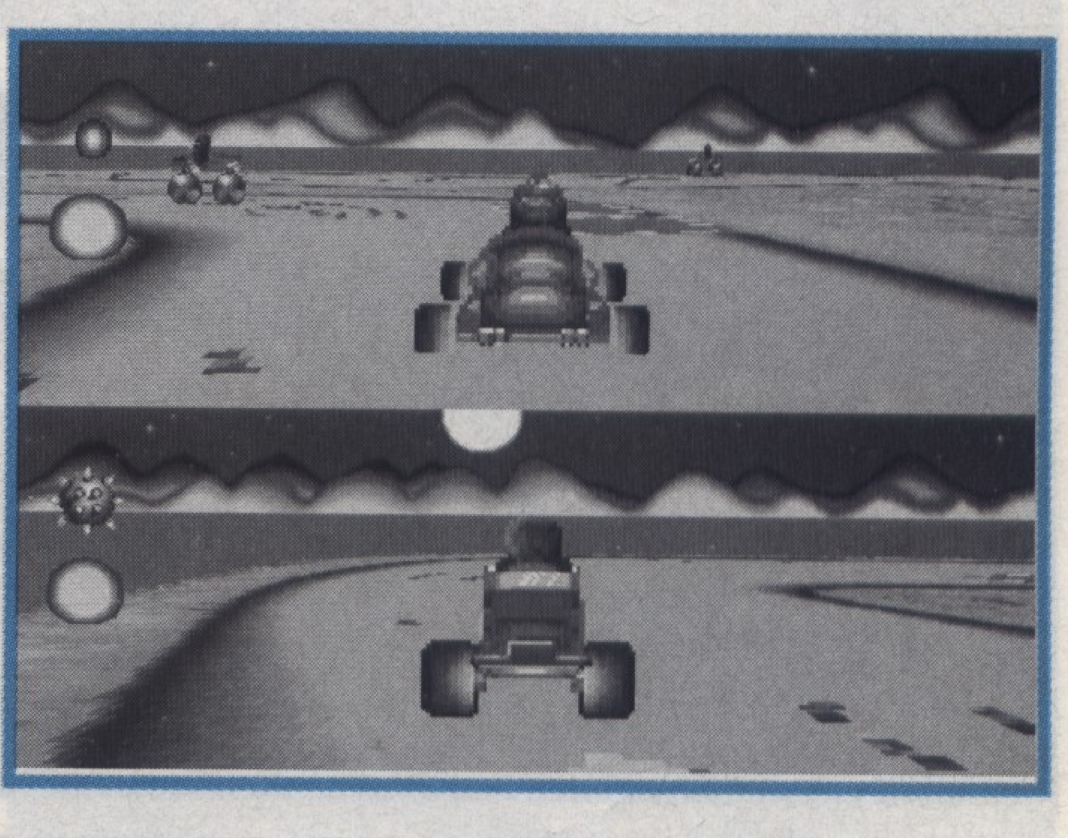
OJOS[]=5,976,980,977,978,979;
O1[]=1,980;
O2[]=1,986;
O3[]=1,417;
O4[]=1,601;
O5[]=1,602;
O6[]=1,603;
O7[]=1,604;
O8[]=1,850;

//DIFERENTES VISTAS DE CADA PERSONAJE CON
LAS RUEDAS "MOJADAS". (CUANDO
//SE PASA POR ENCIMA DE AGUA.)

vpeaso11[]=8, 8, 9, 551, 3, 4, 5, 6, 7;
vpeaso22[]=8, 15, 16, 550, 10, 11, 12, 13,
14;
vpeaso33[]=8,-54, 58, 552, -58, 54, 56, 50, -
56;
vpeaso44[]=8,-55, 59, 553, -59, 55, 57, 51, -
57;
vmono11[]=8,208,209,554,203,204,205,206,2
07;
vmono22[]=8,215,216,557,210,211,212,213,2
14;
vtaja11[]=8,308,309,555,303,304,305,306,30
7;
vtaja22[]=8,315,316,556,310,311,312,313,31
4;
vo[8]=-1;
o[8]=-1;
pr[8];
fu[8];
TIM[8];
vo2[8]=-1;
svol[8];
sd[8];
svol2[8];
sd2[8];
players;
car1;car2;
car3;car4;
die[48]=0;
terres;
carc1;carc2;l1;l2;l3;l4;
carc3;carc4;
cod[8]:=0;
tp[4];
tcpu[4];
dp[4];
```







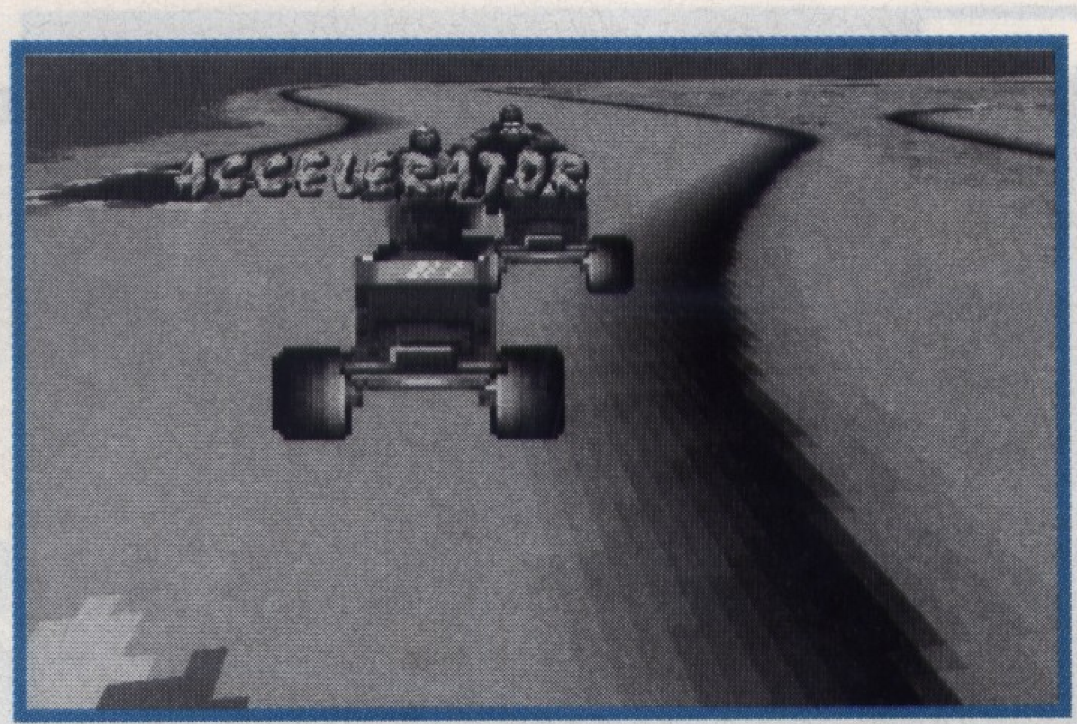
EL JUEGO ADMITE HASTA CUATRO JUGADORES SIMULTANEOS. AQUI HAY UN EJEMPLO DE UNA CARRERA CON DOS JUGADORES.

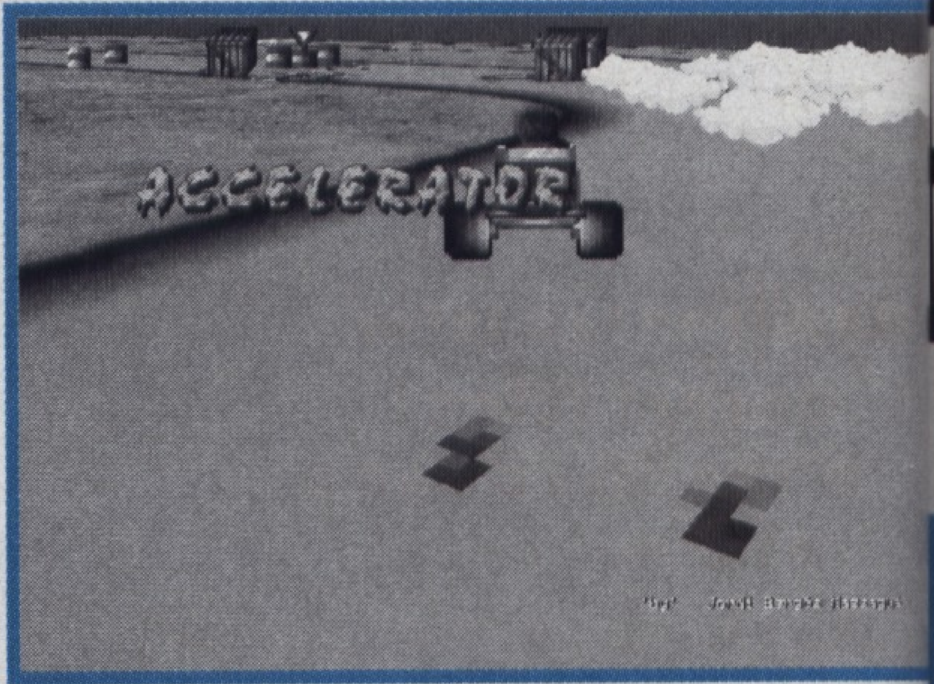
```
dcpu[4];
btp[4];
btcpu[4];
bdp[4];
bdcpu[4];
ALL;
emt;
saigua;
LOCAL

PRIVATE
AG;
JOC;
codi2;
//————————————BEGIN————————————

BEGIN

//ESCOGE DIFERENTES POSIBILIDADES DE
PRESENTACI¢N (REFERENTE A TIPO DE
// KARTS)//
a:=rand(1,4);
if (a==1) car1:=1;car2:=1;
car3:=1;car4:=1;
end;
if (a==2) car1:=2;car2:=2;
car3:=2;car4:=2;
 end;
if (a==3) car1:=3;car2:=3;
car3:=3;car4:=3;
end;
if (a==4) car1:=4;car2:=4;
car3:=4;car4:=4;
end;
//-CARGA DE SONIDOS——————————
```





También existen manchas de aceite que nos hacen perder el control.

```
—————//
strans:=load_pcm("trans.PCM",0);
click6:=load_pcm("click6.PCM",0);
click5:=load_pcm("click5.PCM",0);
click3:=load_pcm("click3.PCM",0);
click4:=load_pcm("abrirmet.PCM",0);
scho33:=load_pcm("choque1.PCM",0);
scho:=load_pcm("a33.PCM",0);

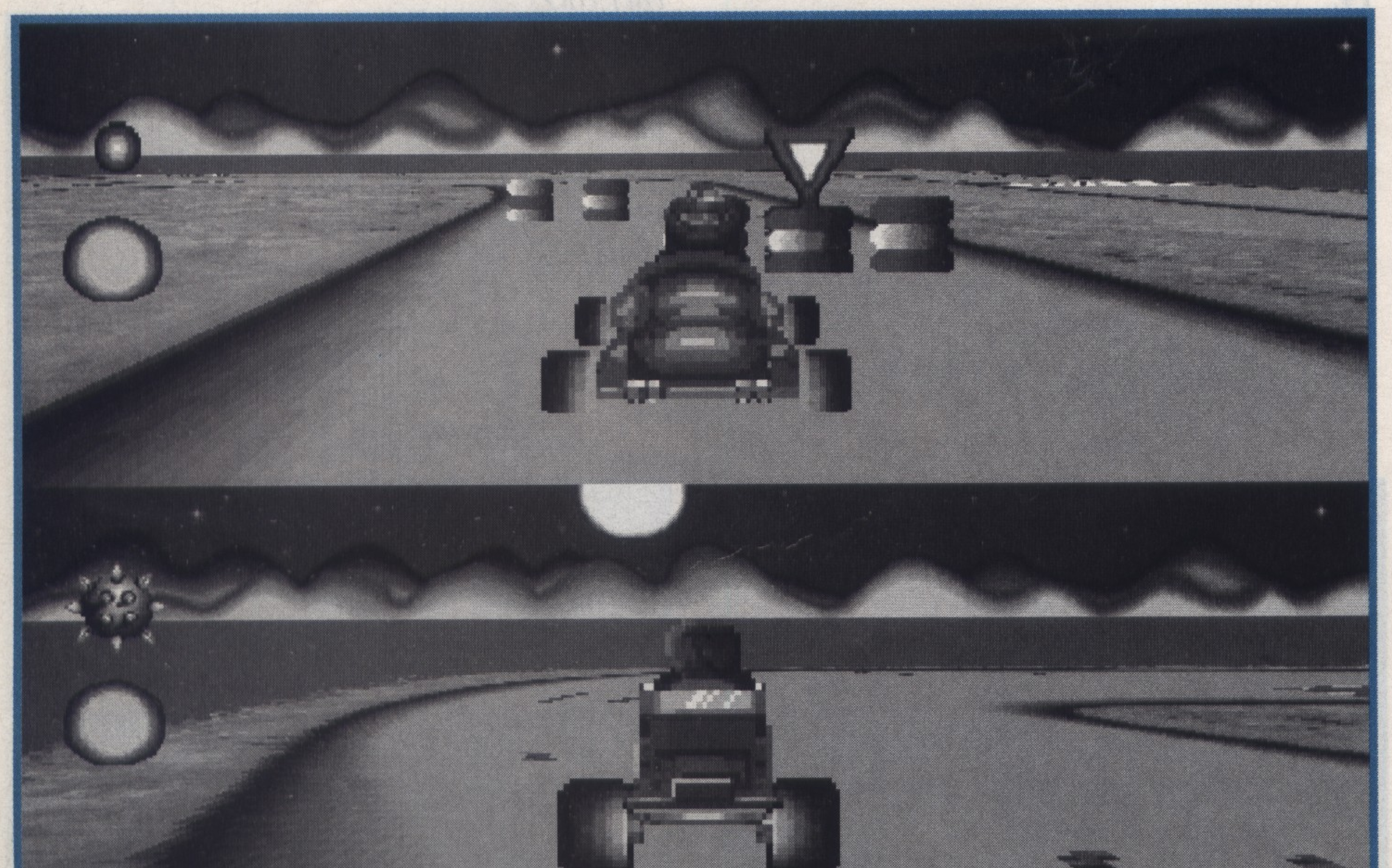
smuelle:=load_pcm("muelle00.PCM",0);
sDERRAPA1:=load_pcm("freno3.PCM",0);
sDERRAPA:=load_pcm("dERRAPA3.PCM",0);
sMOTO:=load_pcm("MOTO.PCM",0);
saigua:=load_pcm("Pez.PCM",0);
sagua:=load_pcm("agua.PCM",0);
sbpe:=load_pcm("bpe.PCM",0);
sbg:=load_pcm("bg.PCM",0);
ssex:=load_pcm("sexplo.pcm",0);
svel:=load_pcm("velo.pcm",0);
spi:=load_pcm("pi.pcm",0);
sapla:=load_pcm("apla.pcm",0);
sapla2:=load_pcm("apla2.pcm",0);
shuy:=load_pcm("uy.pcm",0);
sbp:=load_pcm("bp.pcm",0);
scapsa:=load_pcm("capsa.pcm",0);
scaLA:=load_pcm("cala.pcm",0);
slik:=load_pcm("lik.pcm",0);

//-MENSAJES DEMO——————————————
——————//
 set_mode(m640x400);    //n706

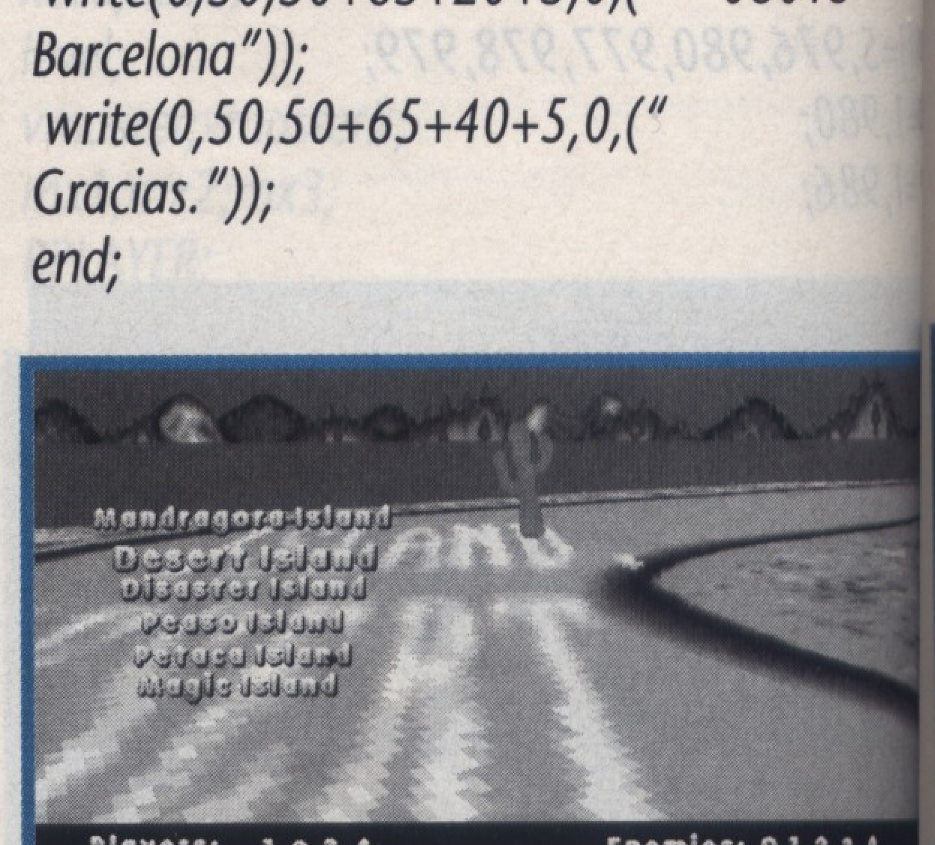
//write(0,50,50,0,("This is a demostration
version."));
//write(0,50,50+10,0,("And you only can play
the first island."));

write(0,50,50,0,("Esta es una versi¢n de
demostraci¢n,"));
write(0,50,50+10,0,("en la cual solo se puede
acceder a la primera isla."));

if (rand(0,1))
 write(0,50,50+30,0,("Si deseas recibir el juego
completo con todas las islas"));
 write(0,50,50+40,0,("solo tienes que enviar
una carta indicando muy claramente tu direcci
"));
 write(0,50,50+50,0,("y 2000 pts, o (20 $)
(20 euros) a: "));
 write(0,50,50+65+5,0,("     Jordi Berg¢s
Massagu,"));
 write(0,50,50+65+10+5,0,("    c\Tamarit 1
2§4"));
 write(0,50,50+65+20+5,0,("    08015
Barcelona"));
 write(0,50,50+65+40+5,0,("
Gracias."));
end;
```



Los obstaculos que encontramos en la carretera deben ser esquivados continuamente.



Autor: Emilio José Jiménez Jiménez

JUEGOS GANADORES: 3º

E. J. Dominó

El tercer ganador de este mes nos ha enviado una versión informática de uno de los clásicos juegos de mesa españoles. Como habréis podido deducir por el título, nos estamos refiriendo al juego del dominó. Dejamos que sea el propio autor el que explique su creación.



E.J Dominó es, como su propio nombre indica, un juego de dominó en el que un jugador humano se enfrenta contra dos jugadores controlados por el ordenador. Los motivos que me llevaron a hacer este juego están ampliamente explicados en el archivo LEEME.TXT. Pero quisiera decir que gracias a él he profundizado un poco en el tema de la Inteligencia Artificial (I.A.) que es de vital importancia en el mundo de los videojuegos de hoy en día. Intentaré por medio de este artículo explicar a grandes rasgos la mecánica del programa.

ESTRUCTURA DEL PROGRAMA

El programa se divide principalmente en tres partes: la presentación, el sistema de menús y el juego, de modo que al ejecutarlo se pone la presentación para después pasar al "BUCLE PROGRAMA" que se encarga de poner el menú y el juego de forma alternativa hasta que se salga al sistema operativo, momento en el que se pone otra vez la presentación en su variante de despedida.

El menú es el primer punto de contacto directo con el juego y donde el usuario puede empezar a tomar decisiones, por lo tanto debe ser un sistema vistoso y eficiente, que cause una buena impresión. El sistema de menús se basa en el proceso *boton()* que se ocupa de:

- Poner el gráfico del botón con su correspondiente texto en pantalla.



- Gestionar los eventos con el ratón devolviendo un valor sobre la variable opción cada vez que se pulsa sobre él. Además cambia el fondo del botón cuando el ratón está sobre él.

Un jugador humano enfrentado a dos jugadores controlados por el ordenador

Por lo tanto, el sistema de menús se divide en cuatro procesos: *jugar()*, *datos()*, *opciones()* y *salir()*, que son los submenús y que controlan cierto número de botones, encargándose de moverlos por la pantalla cuando sea necesario.

EL JUEGO

El juego es el bloque principal del programa. El proceso *juego()* controla todos los aspectos de las partidas mediante dos bucles tipo *LOOP...END*. En el primero de ellos, que llamaremos "BUCLE PARTIDA", controla quién es mano y debe salir cada vez que se comienza una partida. En el segundo, "BUCLE TURNO", se encarga de controlar quién debe poner ficha en cada momento de la partida. Además en este bucle se comprueba el estado de la partida mediante el proceso *comprobar_variables()*, para ver si alguien ha dominado o se ha cerrado. En caso de que la partida haya concluido, este proceso se encarga de determinar quién es el ganador, para ello cuenta los puntos de cada jugador con la función *contador()*, y de poner los textos indicándolo en pantalla para, después, salir del "BUCLE TURNO", actualizar los marcadores y comenzar una nueva partida.

ESTRUCTURAS

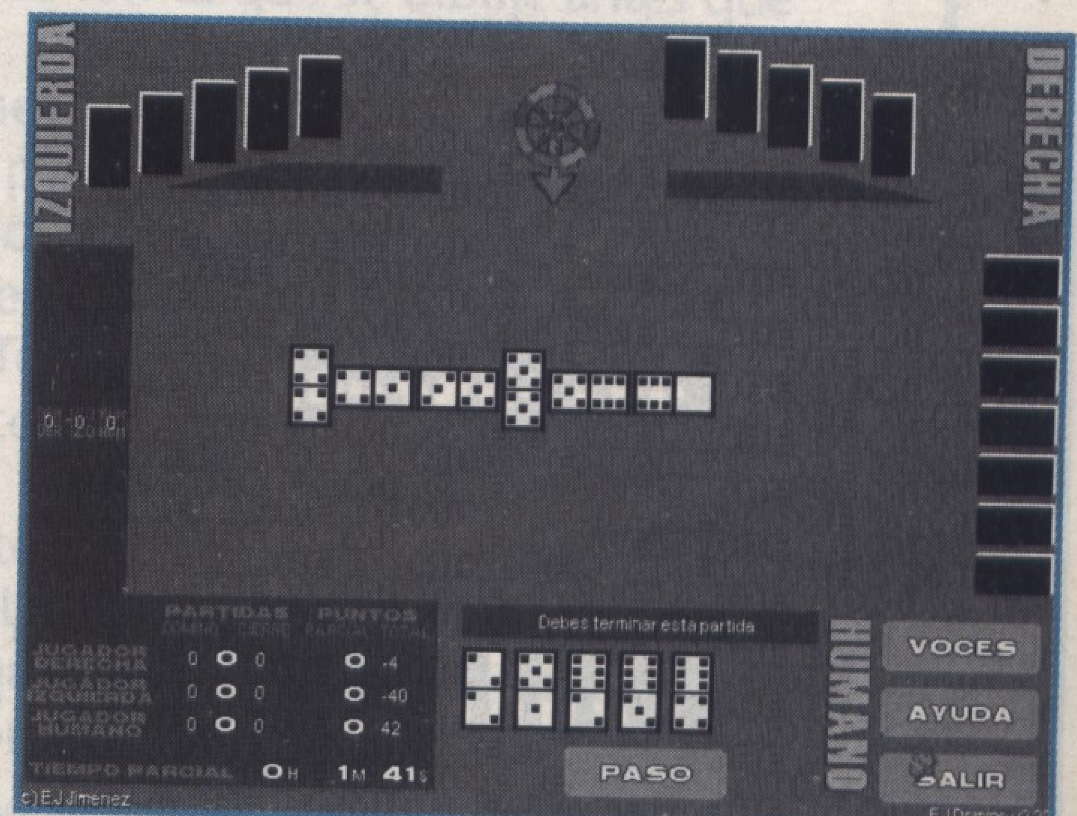
Las más importantes son mesa y jugador[3], pues se encargan de guardar los datos sobre los que luego operarán los algoritmos de puntuación de fichas. Antes de comentarlas, digamos que cada ficha del dominó se compone de dos números que van del 0 al 6, a estos números les llamaremos clases, de esta forma cada ficha está compuesta por dos clases: *c0* y *c1*. Además a cada ficha podemos asignarle un código único haciendo: *codigo=10*c0+c1*, esto es útil a la hora de asociar un gráfico a cada ficha, siendo posible recuperar el valor de *c0* y *c1* a partir del código haciendo: *c1=codigo%10* y *c0=(codigo-c1)/10*.

La estructura mesa agrupa los registros relacionados con las fichas que hay sobre la mesa, los más importantes son:

- *fichas[7,7]*:Se trata de una matriz simétrica binaria que determina que ficha está puesta o no en la mesa. Por ejemplo, si el cuatro-blanca está puesto, su valor sería: fichas[4,0]=fichas[0,4]=1.
- *lado[1].clase*: Es otra estructura que guarda la clase que hay en cada extremo de la mesa.

La estructura jugador. El jugador 0 es el humano y el 1 y el 2 sus contrincantes controlados por el ordenador, el 3 son las fichas que no juegan. Los principales registros son:

- *fichas[7,7]*: Matriz simétrica binaria con las fichas que tiene cada jugador.

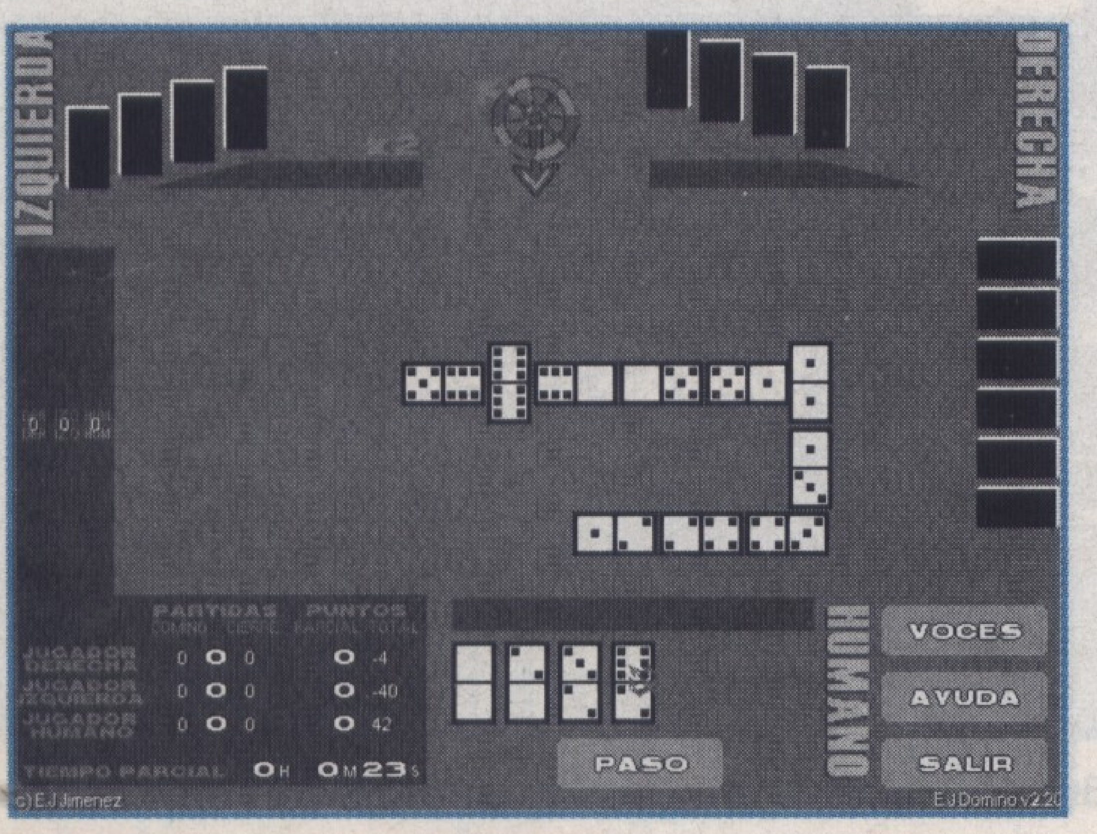


- *n_clase[6]*: Número de fichas de dicha clase que tiene el jugador.
- *clase[6]*: Se trata de un registro con valores de –10 a 10 que nos informa en forma de gradiente si el jugador tiene o no tiene fichas de dicha clase, por ejemplo, si *clase[4]=-10* resulta que el jugador tiene muchos 4, sin embargo si fuera *clase[4]=5* seguramente no tendría ninguno. Este es el gradiente de inferencias que se actualiza a cada ficha que ponga el jugador y que es tenido en cuenta por el algoritmo de búsqueda de fallos. Los demás registros informan sobre los puntos, partidas ganadas, etc. que cada jugador tiene.

JUGADORES

El jugador humano toma las decisiones por sí mismo, luego basta con que comunique al programa qué ficha quiere poner para que éste compruebe que realmente se puede poner y la ponga. El jugador humano pulsa con el ratón sobre la ficha que quiere poner, entonces se accede a un mapa de durezas que nos dice en qué posición está colocada la ficha. Sólo nos queda determinar el código de dicha ficha que se consigue mediante el proceso *buscar_en_matriz()*, al cual se le pasa la posición *p* de la ficha y recorre la matriz de fichas del jugador devolviendo el código de ficha correspondiente al *p-ésimo 1* encontrado. Entonces esa es la ficha que se pone en caso de que se pueda poner.
Los jugadores virtuales básicamente son los homónimos al jugador humano pero controlados por el ordenador. En cada turno se encarga de lo siguiente:

1. Buscar las fichas que se pueden poner en mesa: mediante el proceso *buscar_candidatas()* los códigos de las fichas candidatas a ser puestas son almacenados en la estructura *candidatas*.
2. Puntuar las fichas que se pueden poner en mesa: el proceso *puntuar_candidatas()* se encarga de pasar los códigos de fichas de la estructura *candidatas* de uno en uno a los algoritmos de I.A. que les otorgan una puntuación en función de la conveniencia o no de poner la ficha en mesa.



3. Elegir la ficha que se pondrá en mesa: el proceso *elegir_ficha()* ordena los códigos de las fichas candidatas según su puntuación y se toma aquel con puntuación más alta, entonces se llama al proceso *poner_ficha()* con dicho código como parámetro y se termina el turno para el jugador.

MOTOR DE INTELIGENCIA ARTIFICIAL

La I.A. del juego se basa en asignar una puntuación a cada ficha de modo que la que más puntos obtenga sea la mejor opción a la hora de poner una ficha en la mesa. Para calcular la puntuación se utilizan básicamente dos algoritmos a los que se le pasa el código de ficha (que es descompuesto en *c0* y *c1*), el jugador que la tiene (para poder operar en función de los datos de este jugador), y el nivel del jugador (que es más o menos la destreza con la que juega).

- *algoritmo_basico()*: Puntúa las fichas siguiendo los criterios básicos de: "ESTAR A MI JUEGO", que se aplica sobre *c0* y consiste en no matar fichas de las que tenga mayoría; e "INTENTO DE DOMINIO", que se aplica sobre *c1* y consiste en mantener en los extremos de la mesa las fichas de las que tengo mayoría.

La I.A. del juego se basa en asignar puntos a las fichas y elegir la suma más alta

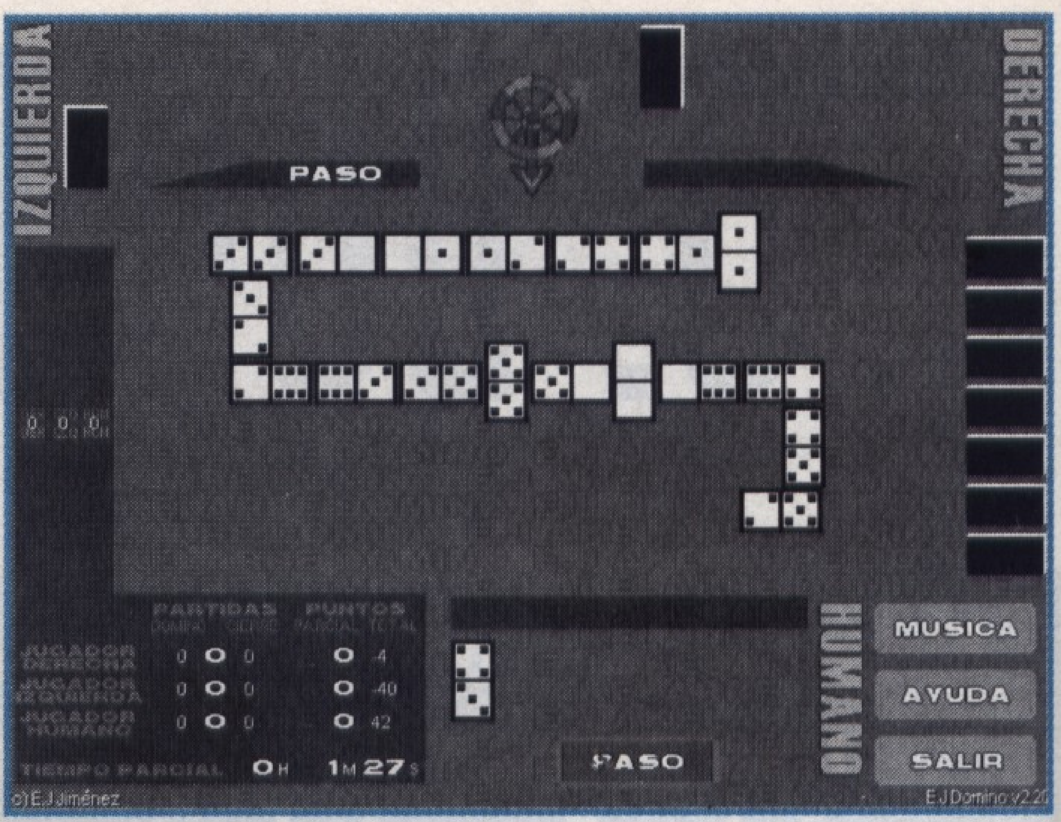
- *algoritmo_fallos()*: Puntúa las fichas atendiendo al criterio de "BUSQUEDA DE FALLOS", es decir, basándose en el gradiente de inferencias de un determinado jugador, establece la posibilidad de que dicho jugador falle a una clase en concreto, puntuando en consecuencia según convenga o no poner dicha clase. Nótese que este algoritmo se aplica tanto a *c1* (clase que se pondrá como nuevo extremo de la mesa) como a *c0* (clase que se "matará" en el extremo), pues es interesante no quitar de los extremos de la mesa fichas a las que cierto jugador falla.

Además existen otros tres algoritmos que son:

- *algoritmo_cambiar()*: Se aplica sobre el global de las fichas para ver si merece la pena jugar con ellas o cambiarlas.
- *algoritmo_salir()*: Cuando el jugador es mano determina cuál es la mejor ficha para salir.
- *algoritmo_cierre()*: Cuando al poner cierta ficha se puede cerrar, dice al jugador si tiene posibilidades de ganar el cierre.

PROCESOS AUXILIARES

Si miras el código fuente verás gran cantidad de procesos que no he nombrado pero que son de vital importancia para el correcto desarrollo del juego. Los principales son:



- *poner_ficha()*: Recibe como parámetro el código de la ficha que pondrá en pantalla, encargándose de ponerla en el lugar preciso. Después actualiza las variables y matrices correspondientes para que reflejen los cambios ocasionados al poner en la mesa la ficha.
- *contador()*: Recibe como parámetro una clase, un lugar y un jugador. Recorre la matriz de fichas determinada por el parámetro jugador y por el parámetro lugar en busca de fichas que contengan la clase indicada devolviendo cuantas hay.
- *repartir_fichas()*: Se encarga de repartir aleatoriamente las fichas cada vez que empieza una partida.
- *pintar_fichas()*: Pinta en pantalla las fichas de un jugador determinado. Sirve para actualizar la imagen en pantalla cuando un jugador pone una ficha y por lo tanto deja de tenerla.

actualizar_marcadores(): Pone los puntos en los marcadores de uno en uno y actualiza las barras de puntos.
cuenta_puntos(): Cuenta el número de puntos que tiene un jugador recorriendo su matriz de fichas de forma que cada vez que encuentra un 1 suma a un *contador_de_chinos* las coordenadas donde se ha encontrado el 1.
cargar_salvar(): Se encarga de la gestión de las operaciones que hay que hacer antes de cargar y después de salvar.
construir_fondo(): Los marcadores del tablero de juego se construyen mediante *tiles* para ahorrar espacio en disco.
tracker(): Se encarga de "pinchar" aleatoriamente la música del juego.
Y un largo etc. que ni yo mismo sería capaz de enumerar de memoria. Espero que estas superficiales explicaciones, junto con el código fuente, te ayuden a comprender mejor el funcionamiento del juego. No tengo correo electrónico pero si quieres ponerte en contacto conmigo para intercambiar ideas en torno a *DIV* y a la programación de videojuegos escríbeme a la dirección que figura en el archivo LEEME.TXT del juego. Un saludo.

- Watcom C++: los primeros 4 parámetros se reciben en los registros EAX, EDX, EBX, ECX de la CPU por este orden, y si hay alguno más, se depositan en la pila ordenados de izquierda a derecha.
- Borland C++: los primeros 3 parámetros se reciben en los registros EAX, EDX y ECX de la CPU por este orden, y si hay alguno más, se depositan en la pila ordenados de derecha a izquierda. Es igual al método de *Watcom* excepto que conserva por alguna razón el valor del registro de EBX, además de invertir el orden de la pila.
- Visual C++: los 2 primeros parámetros se depositan en los registros ECX y EDX, depositándose, si hay alguno más, en la pila del sistema de derecha a izquierda.

DIV tan solo reconoce el formato de toma de parámetros de *Watcom* (fastcall), por lo que si deseamos utilizar nuestras funciones desde otro compilador, debemos hacer las modificaciones pertinentes para que nos sea reconocida la DLL. Para ello debemos redefinir las funciones principales externas como son COM_export y div_text_out, de forma que alteremos todos los registros al método *watcom*. Para ello vamos a diseñar dos nuevas funciones que las sustituyan como son BC_COM_export y BC_div_text_out. La primera difiere de la original en que recibe 4 parámetros, siendo el primero la dirección de la función COM_export y los otros 3 los parámetros normales de COM_export. Por tanto, los pasos a realizar se pueden ver en el cuadro 2 (pseudocódigo).

Pasando este pseudocódigo a ensamblador, quedaría tal que:

```
BC_COM_export PROC
funcion,para1,para2,para3
  PUBLIC BC_COM_export
   Pushad
   Mov ebx,para3
   Mov edx,para2
   Mov eax,para1
   Mov ecx,funcion
   Call ecx
   Popad
   Ret
  BC_COM_export ENDP
```

El mismo código sería válido para la función *BC_div_text_out*, variando el valor de la función almacenado en ECX por un puntero al código de *div_text_out*.

El último problema que se nos presenta ahora es el tema del tratamiento de ficheros. Pero antes, veamos cómo debemos manipular archivos en las DLLs para *Watcom*.

Manipulación de datos por archivo

A la hora de acceder a un archivo en las bibliotecas de *DIV*, debemos tener siempre en cuenta que se hace sin ningún tipo de restricción



Muestra del funcionamiento de la biblioteca para cargar archivos PCX de n0c.

exceptuando la de utilizar las funciones *div_fopen* y *div_fclose* para abrir y cerrar archivos. Estos son sus prototipos:

```
FILE * div_fopen (char*,char*);
void div_fclose (FILE *);
```

Veamos un ejemplo de cómo se lee el primer byte de un archivo binario:

```
//..
FILE fichero;.
Byte temp;
fichero =
div_fopen(archivo_a_abrir,"rb");
if (fichero != NULL)
{
  fread(&temp,1,1,fichero);
  div_fclose(fichero);
}
```

El problema de la manipulación de ficheros en otros compiladores

Surge un último problema cuando intentamos acceder a las funciones de manipulación de archivos de

Cuadro 1. Código fuente de la dll de lectura de PCX.

Tal vez lo único que pueda parecer extraño sea la sentencia de carga del nombre del archivo PCX en la función load_pcx. Como dijimos en el anterior número, existe un offset o desplazamiento dentro de la memoria de DIV donde se encuentran todos los textos. Se puede acceder a esta dirección mediante la sentencia mem[text_offset]. Cuando enviamos una cadena a una función, se nos envía como parámetro un incremento de desplazamiento dentro de ese offset, de forma que basta con sumarlo a la dirección anterior para encontrar el puntero al principio de la cadena con el nombre del fichero.

Como podéis observar, esta es una versión muy preliminar en la que tan sólo se cargan imágenes con un tamaño de 320x200 pixeles. No obstante el lector puede hacer muchas más mejoras como las que citamos a continuación:

Hacer comprobaciones del formato PCX, ya que no se verifica en este breve código.

Adaptar el código de forma que cargue imágenes de cualquier tamaño. Para ello se deben usar las variables de entorno width y height que nos indican el ancho y alto de la pantalla respectivamente, de forma que podamos encuadrar bien la imagen en la pantalla

Como vemos, al realizarse el volcado de imagen después de procesarse el buffer de imágenes (post_process_buffer), la imagen PCX siempre aparecerá encima de cualquier proceso. Si queremos que aparezca detrás de todos los procesos que se dibujen debemos cambiar tan sólo el nombre de la función por background_to_ buffer de forma que se dibuje antes que todos los procesos restante.

En cada frame se abre y cierra el fichero PCX, lo cual puede ralentizar excesivamente el programa. Por ello es conveniente hacer una copia del fichero en memoria, comprimido o sin descomprimir, para que tan sólo debamos hacer un volcado de memoria, que es mucho más rápido que desde fichero. Al hacer esa copia en la función load_pcx, evitamos también mucha de la carga del programa. Es importante tener en cuenta que para liberar la memoria con la función div_free(), se debe hacer en la función divend, no presentada hasta ahora. Todo dato, fichero o puntero que deba ser liberado cuando termine el programa se debe hacer con esta función, así como también se permite inicializar variables y hacer todo tipo de operaciones de inicialización en la función divmain.

Esto es todo por hoy. Os animo a que experimentéis un poco con el código fuente y veáis las múltiples opciones que nos ofrece. También os recomiendo que consultéis el código fuente de HBOY.DLL, que acompaña a DIV, ya que aglutina conceptos como la descompresión RLE, manejo de paletas y otros muchos que le pueden ser de mucha ayuda.

Para cualquier tipo de consulta, podéis remitir un correo electrónico a la dirección trinidad@arrakis.es.

Cuadro 2. Proceso de conversión de la función COM_export.

Borland C++	Visual C++

Almacenar en la pila del sistema los valores de todos los registros de la CPU
Colocar en EAX el valor del segundo parámetro (primero de COM_export)
Colocar en EDX el valor del tercer parámetro (segundo de COM_export)
Colocar en EBX el valor del cuarto parámetro (tercero de COM_export)+Colocar en ECX el valor del primer parámetro, que es un puntero a la función de Watcom
Llamar a la función situada en ECX
Una vez terminada la ejecución de la función restaurar los valores de los registros de la CPU almacenados en la pila

otros compiladores. Esto se debe al distinto formato utilizado por cada compilador en la estructura FILE. En esta estructura tenemos diversos datos, como un puntero al siguiente *byte* a leer, número de *bytes* al final del fichero y lo más importante, un *handle* o manipulador que identifica al archivo abierto internamente. Puedes ver la estructura de FILE en dos entornos distintos, como son *Watcom* y *Visual* C++ en la cuadro 3.

Por esta razón, debemos usar siempre en las DLLs de *DIV* la estructura FILE de *Watcom*, con lo que aparece ya una tercera y última diferencia a solventar. La solución nos la ofrece el lenguaje ensamblador. *Borland C++* y *Visual C++* nos permiten crear directamente código ensamblador sobre el código C o añadirlo en forma de archivo LIB precompilado. Para ello recurrimos a la interrupción 21h que nos permite acceder a ficheros. Para ver los parámetros que reciben las correspondientes subfunciones de la interrupción véase la cuadro 4.

Veremos un ejemplo de cómo podemos escribir una función que emule el comportamiento de fwrite en ensamblador externo

Veamos un ejemplo de cómo podemos escribir una función que emule el comportamiento de fwrite en ensamblador externo, teniendo en cuenta que recibe como parámetro WFILE (FILE de *Watcom*) en vez de la estructura FILE:

```
BC_fwrite PROC
direc,size,num,mifile:near ptr wfile
    PUBLIC BC_fwrite
    Push ebx
    Push ecx
    Push edx
    Mov ecx,size
    Imul ecx,num
;multiplica el tamaño de los datos
por su número para obtener el
número de bytes
    Mov ebx,mifile
    Mov ebx,[ebx+16] ; desplazamiento del handle dentro de la
estructura FILE de Watcom.
    Mov ax,4000h
    Mov edx,direc
    Int 21h
    Xor edx,edx
    Div size ; div divide EAX entre
la variable size para devolver el
número de datos escritos.
    Pop edx ; restaura los valores
anteriores de los registros de la CPU
    Pop ecx
    Pop ebx
    Ret
    BC_fwrite ENDP
```

Llegando a una conclusión.

Es obvio que si cada vez que quisieras escribir una DLL en *Borland* C++ tuvieses que hacer todo esto, sería un fastidio. Por ello, vamos a resumir los pasos, después de tanta teoría, que debemos realizar para compilar nuestra DLL. Necesitarás el fichero BC50DLL.zip que puede encontrar en el CD o en varios lugares de la *web*, entre ellos el FTP de *DIV* alojado en la página *web* de Deemo (pagina.de/deemo). Además debes tener en cuenta que sólo funcionará en compiladores *Borland* de 32 bits, como son las versiones 4.5x en adelante.

Cuadro 3. Diferencias del tipo de dato FILE en Watcom y Visual C++.

Watcom C++

```
Struct FILE{
    Unsigned char *_ptr;
    Int _cnt;
    void *_base;
    unsigned _flag;
    int _handle;
    unsigned _buffersize;
    unsigned short _ungotten;
}
```

Visual C++

```
Struct FILE{
    Char *_ptr;
    Int _cnt;
    Char *_base;
    Int _flag;
    Int _filehandle;
    Int _charbuf;
    Int _bufsize;
    Char *_tempfname;
}
```



Crea un proyecto con las opciones que ve en el cuadro 5.

Elimina el archivo de recursos *.RC

Añade un archivo DEF al proyecto

Añadir el archivo BCDIV.LIB al proyecto incluido en el paquete anteriormente citado.

Añadir la línea #include "bcdiv.h" después de #include "div.h" en el archivo cpp de nuestro código.

Ten en cuenta la necesidad de las funciones BC_div_text_out y BC_COM_export.

Es recomendable usar variables globales inicializadas, ya que si no, se almacenan en la sección BSS, aparentemente limitada en *DIV* a 300 bytes, con lo que al inicializarse pasarían a la sección DATA, no limitada.

La extensión del fichero de código, debe ser CPP aunque no utilicemos las opciones que nos ofrece la programación orientada a objetos de C++ ni ninguna de sus capacidades.

Las Paletas de DIV

Volviendo al tema de la descompresión de archivos PCX que nos ocupaba en el anterior número, veamos unos últimos conceptos acerca de las paletas de colores. Podemos decir que *DIV* posee dos tipos de paletas, cada una representada por:

- *Palette:* es la paleta del juego. Durante la ejecución de un programa no se altera, a menos que carguemos otra o que la modifiquemos mediante alguna función en una dll. Para ello debemos hacerlo en la función *process_palette(void)*.

• *Active_palete:* es la paleta actual con la que se presentan los gráficos. Se utiliza sobre todo para efectos de *fading* o fundidos, que varían de forma temporal la paleta del juego hasta que se termina el efecto, de forma que para volver a la normalidad basta con copiar la paleta del juego *(palette)* sobre ésta. Para modificarla debemos recurrir a la función *process_active_palette(void)*.

En consecuencia, usaremos la dirección *palette* para cargar la paleta de nuestro archivo PCX, para así permitir efectos de fundido y otros.

Con esto ya tenemos todo lo necesario para terminar el desarrollo de nuestra librería de lectura de archivos PCX. Para terminar de desarrollarla, y aprovechando la explicación de cómo compilar con *Borland C++* vamos a analizar la librería desarrollada por n0c que amablemente nos ha cedido, realizada para este entorno. Para realizar la traducción a *Watcom*, basta con eliminar en todas las funciones *BC_x*, las letras *BC_* y en *BC_COM_export* eliminar el primer parámetro, dejando tan sólo tres. El código fuente se encuentra en el Cuadro 1, aunque también podéis verlos en el CD que acompaña a esta edición.

Tal vez lo único que pueda parecer extraño sea la sentencia de carga del nombre del archivo PCX en la función *load_pcx*. Como dijimos en el anterior número, existe un *offset* o desplazamiento dentro de la memoria de *DIV* donde se encuentran todos los textos. Se puede acceder a esta dirección mediante la sentencia *mem[text_offset]*. Cuando enviamos una cadena a una función, se nos envía como parámetro un incremento de desplazamiento dentro de ese *offset*, de forma que basta con sumarlo a la dirección anterior para encontrar el puntero al principio de la cadena con el nombre del fichero.

Como podéis observar, esta es una versión muy preliminar en la que tan sólo se cargan imágenes con un tamaño de 320x200 pixeles. No obstante el lector puede hacer muchas más mejoras como las que citamos a continuación:

Hacer comprobaciones del formato PCX, ya que no se verifica en este breve código.

Adaptar el código de forma que cargue imágenes de cualquier tamaño. Para ello se deben usar las variables de entorno *width* y *height* que nos indican el ancho y alto de la pantalla respectivamente, de forma que podamos encuadrar bien la imagen en la pantalla

Cuadro 4. La interrupción 21h y sus subfunciones de manejo de fichero.

Interrupción 21h

Función	Lectura	Escritura	Mover puntero
EAX	3F00h	4000h	420xh*
EBX	Handle del fichero	Handle del fichero	Handle del fichero
ECX	Bytes a leer	Bytes a escribir	
EDX	Dirección de los datos	Dirección de los datos	Incremento
Devuele	EAX = Bytes leídos	EAX = Bytes escritos	CF = 1 error, CF = 0: EAX puntero a fichero

* el valor de x depende de la posición base:
SEEK_SET(0),SEEK_CUR(1),SEEK_END(2)

Como vemos, al realizarse el volcado de imagen después de procesarse el buffer de imágenes *(post_process_buffer)*, la imagen PCX siempre aparecerá encima de cualquier proceso. Si queremos que aparezca detrás de todos los procesos que se dibujen debemos cambiar tan sólo el nombre de la función por *background_to_buffer* de forma que se dibuje antes que todos los procesos restante.

En cada frame se abre y cierra el fichero PCX, lo cual puede ralentizar excesivamente el programa. Por ello es conveniente hacer una copia del fichero en memoria, comprimido o sin descomprimir, para que tan sólo debamos hacer un volcado de memoria, que es mucho más rápido que desde fichero. Al hacer esa copia en la función *load_pcx*, evitamos también mucha de la carga del programa. Es importante tener en cuenta que para liberar la memoria con la función *div_free()*, se debe hacer en la función *divend*, no presentada hasta ahora. Todo dato, fichero o puntero que deba ser liberado cuando termine el programa se debe hacer con esta función, así como también se permite inicializar variables y hacer todo tipo de operaciones de inicialización en la función *divmain*.

Esto es todo por hoy. Os animo a que experimentéis un poco con el código fuente y veáis las múltiples opciones que nos ofrece. También os recomiendo que consultéis el código fuente de HBOY.DLL, que acompaña a *DIV*, ya que aglutina conceptos como la descompresión RLE, manejo de paletas y otros muchos que le pueden ser de mucha ayuda.

Para cualquier tipo de consulta, podéis remitir un correo electrónico a la dirección trinidad@arrakis.es.

Pablo Trinidad

Cuadro 5. Opciones de un proyecto de DLL en Borland C++.

Opciones de compilación al crear la DLL

Target Type	Dynamic Library [.dll]
Platform	Win32
Target Model	GUI
Standard Libraries	Runtime (desmarca OWL y Class Library)

Si se dispone del Borlan Power Pack para DOS, también se pueden crear como Platform

Platform	DOS (32 bit DPMI)
Target Model	Console
Standard Libraries	Runtime Static

En las opciones del proyecto (Options/Proyect o Edit local options)
32-bit Compiler

Processor	Instruction set: 80386 (o superior)
	Data alignment: Double word (4-byte)
Calling Convention	Register

Direct X
Configuración de Visual C++

Aunque parezca una nimiedad, si no configuramos adecuadamente el Visual C++ no podremos compilar nuestro código. Normalmente no solemos prestar mucha atención a los ficheros Readme, esta vez es un grave error.

Entre la documentación que nos presenta Microsoft, se nos advierte que debemos poner en primer lugar las librerías de Microsoft para poder compilar nuestro código correctamente.

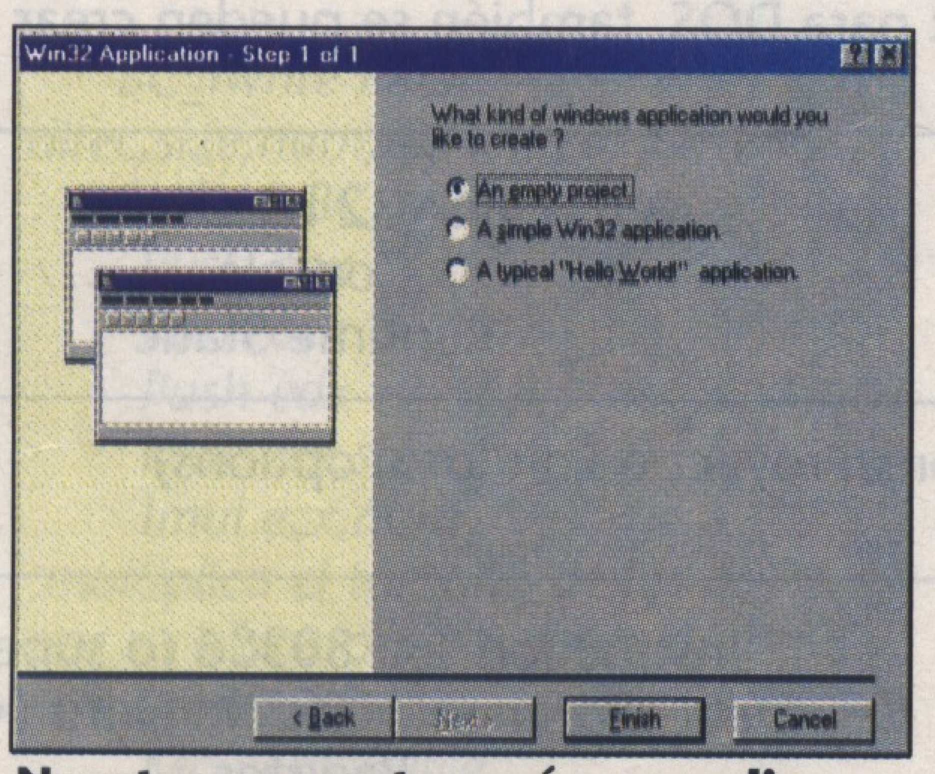
Debemos pues, incluirlos entre los directorios de *librerías* y de *includes*, los que tenemos en el SDK de DirectX. Si no lo hacemos de esta forma, no podremos trabajar.

DirectMedia

Si pensábamos que con DirectX teníamos suficiente para realizar todo aquello que necesita de gran potencia, podemos encontrarnos con un obstáculo insalvable. Habitualmente usábamos las rutinas de la API de Windows para reproducir vídeo, sin embargo, si necesitamos reproducir vídeo de alta calidad y a pantalla completa, veremos que las API no son capaces de alcanzar el alto número de *frames* que necesitamos. Este es uno de los problemas que resuelven las DirectMedia.

Microsoft ha creado un conjunto de rutinas para reproducir multimedia con gran calidad, aprovechando la potencia de las DirectX, y nos proporciona nuevas formas de trabajo que dan la velocidad que con las API no podíamos conseguir.

Aunque no tengamos intención de programar bajo DirectX, la instalación de SDK de DirectMedia, es altamente recomendable para aquellos que habitualmente programen aplicaciones multimedia y necesiten gran potencia en sus aplicaciones.



Nuestro proyecto será una aplicación Win32 vacía.

Instalación de DirectX

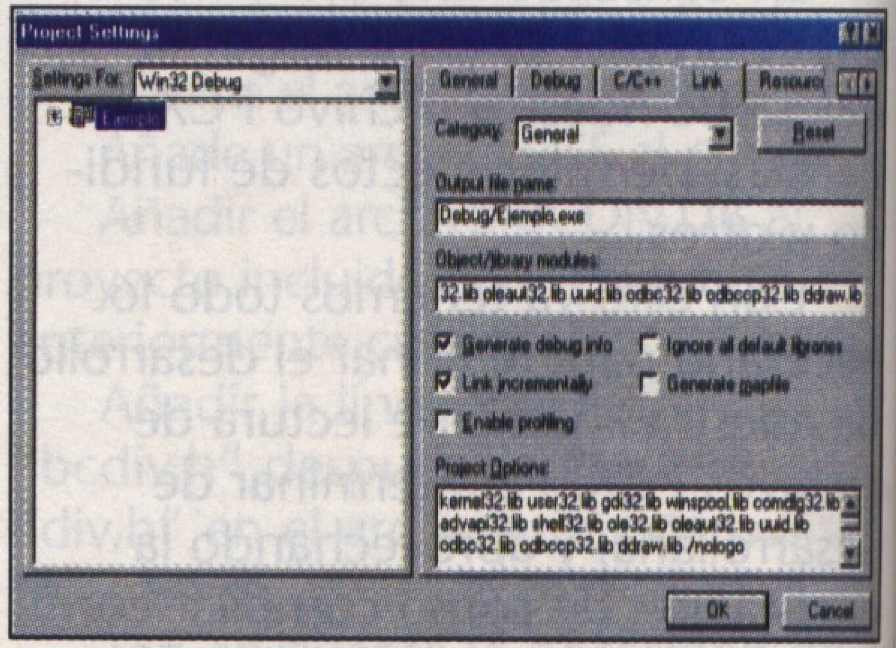
La instalación de DirectX es bastante sencilla, debemos elegir entre la instalación completa o la reducida. Personalmente recomiendo la instalación completa, ya que contiene un gran número de ejemplos que pueden sernos de gran utilidad.

Otra de las opciones que se nos presenta es la de instalar la versión *Retail* o la *Debug*. Nuestra elección debe ser *Debug*, ya que ésta nos proporciona una mayor información a la hora de depurar nuestro código, perdemos velocidad cuando queramos jugar, pero ganamos en la programación. En cualquier caso, no apreciaremos una merma de la calidad en nuestros juegos habituales.

Una recomendación para la instalación, cuanto más corto sea el *path* en el que instalemos el SDK, mejor. Debemos tener en cuenta que exploraremos habitualmente el directorio del SDK, al menos al principio. Los ejemplos si no son maravillosos, sí son ilustrativos de lo que podemos hacer con las DirectX.

Requisitos para la programación

Si somos uno de los que tenemos animadversión a las matemáticas, y queremos programar las Direct3D, más vale que nos pongamos al día. Las Direct3D son pura geometría. No necesitamos ser unos grandes matemáticos, pero debemos tener unas nociones básicas de trigonometría, así como de calculo matricial. Esto nos ayudará en nuestro trabajo.



Debemos incluir entre las opciones de nuestro proyecto, la librería ddraw.lib para poder linkarlo correctamente.

Programar las DirectX supone tener un conocimiento bastante avanzado de C++. Conocer este lenguaje, y dominar la programación orientada a objetos, es de carácter obligado para aprovechar las posibilidades de estas librerías.

Depurar el código

Debemos tener en cuenta, antes de comenzar nuestro desarrollo, si vamos a mostrar nuestros gráficos en modo ventana o a pantalla completa. No es una elección sencilla, aparentemente programaríamos nuestro código pensando en trabajar a pantalla completa, que al fin y al cabo va a ser lo que el usuario va a ver. Pero también tenemos que pensar en los diferentes aspectos que la programación nos ofrece. DirectX tiene características diferentes en modo ventana y en modo a pantalla completa, por lo que contemplar ambos aspectos sería demasiado arduo. Sin embargo, hay algo que hará que nos decantemos por la opción en modo ventana. Con la opción a pantalla completa no podemos depurar el código. Si ya es difícil de por sí depurar el código, a pantalla completa es prácticamente imposible. Pero esto si que podemos hacerlo en modo ventana, aun así, debemos ser precavidos, ya que determinadas instrucciones bloquearán nuestra pantalla, obligándonos a resetear.

Nos decantemos por una u otra forma de trabajo, una de las mejores formas de obtener información de depuración es la de generar un fichero de texto con la información

que necesitemos, ésta será la única forma fiable de obtener datos de lo que pasa con nuestro código, por lo que la planificación de un buen sistema de depuración nos proporcionará una información insuperable.

El futuro de DirectX

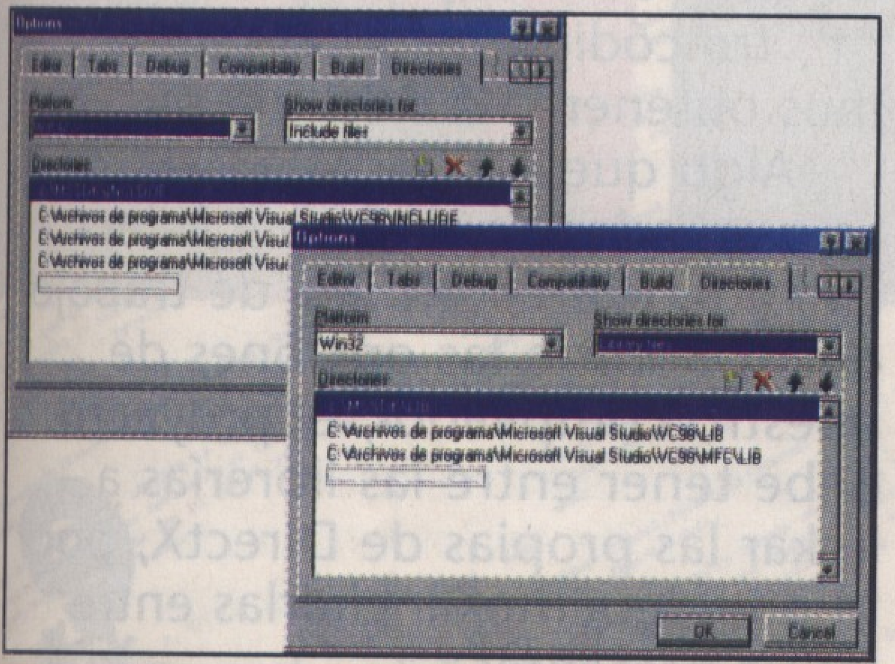
¿Cuál es el éxito de DirectX? Su arquitectura, esa es su principal ventaja. DirectX logrará que la tarjeta gráfica haga todo el trabajo sucio, y si ésta no puede con lo que le pedimos, DirectX lo emulará con software, aunque a costa de una merma de rendimiento considerable.

Aunque, no son aceptadas por todos los programadores fácilmente, es inevitable su uso actualmente. Preferiríamos utilizar otra librería. *Fahrenheit* es el futuro que pretende imponernos Microsoft, la alianza en el desarrollo de esta librería con Silicon Graphics augura un buen futuro para esta nueva librería. Aunque aún debe trabajar mucho Microsoft para sorprendernos gratamente.

Ejemplo sencillo

Pasemos a ver un sencillo ejemplo de programación con DirectX. Antes de intentar inicializar las DirectDraw, debemos preguntarnos que tenemos en la máquina en la que ejecutamos nuestro programa, y para eso, tenemos la función DirectDrawEnumerate. Esta función es de tipo *callback*, y es algo especial. Debemos crear una función de enumeración que será la que se llamará DirectDrawEnumerate. De esta forma, en sucesivas llamadas que el sistema realizará automáticamente, obtendremos toda la información necesaria para llamar a la función de creación del objeto DirectDraw.

Una de las funciones que debemos utilizar en primer lugar, es DirectDrawCreate. Esta función nos permite crear un objeto DirectDraw para trabajar con la tarjeta de vídeo. Podemos pensar que si vamos a programar las Direct3D, no necesitaremos conocer las



Los directorios lib e include deben estar entre los directorios de nuestro compilador, y siempre en primer lugar.

Código fuente del ejemplo

```
#define INITGUID

#include <windows.h>
#include <windowsx.h>
#include <stdio.h>

#include "ddraw.h"

LPDIRECTDRAW        lpDD;
LPDIRECTDRAW2       lpDD2;

HWND hwnd;
HWND hlistbox;

bool WINAPI EnumDDrawDevice(GUID FAR *lpGUID, LPSTR lpDDescription, LPSTR lpDName, LPVOID lpContext)
{
  LONG    indice;
  HWND    hwnd;
  LPVOID  lpDevice;

  // Guardamos el handle que le hemos pasado como información adicional
  // El handle corresponde a una lista

  hwnd = (HWND) lpContext;

  // Guarda en la lista el nombre del dispositivo

  indice = SendMessage(hwnd, LB_ADDSTRING, 0, (LPARAM) lpDDescription);

  // Si hemos tenido algun error, le decimos a la función de enumeración que hemos acabado.

  if (indice == LB_ERR)
     return DDENUMRET_CANCEL;

  // Si GUID es NULL, tenemos el dispositivo primario,
  //        en caso contrario, tenemos otro dispositivo y debemos guardar sus datos.

  if (lpGUID == NULL)
     lpDevice = NULL;
  else
  {
     // Reclamamos memoria para guardar la información

     lpDevice = (LPGUID) malloc(sizeof(GUID));

     // Comprobamos que realmente hemos obtenido la memoria

     if (!lpDevice)
       return false;

     // Copiamos los datos

     memcpy(lpDevice, lpGUID, sizeof(GUID));
  }

  // Guardaremos en la listbox junto al texto, la dirección del dispositivo

  SendMessage(hwnd, LB_SETITEMDATA, indice, (LPARAM) lpDevice);

  // Permitimos que Windows siga llamando a la función mientras tengamos más dispositivos

  return DDENUMRET_OK;
}

// Controla los mensajes de nuestra ventana

LRESULT CALLBACK WindowProc(HWND hwnd, unsigned Msg, WPARAM wParam, LPARAM lParam)
{
  LPGUID lpDevice;
  LONG indice;

  switch(Msg)
  {
    case WM_COMMAND:
      if ((HWND) lParam == hlistbox)
        if (HIWORD(wParam) == LBN_DBLCLK)
        {
          lpDevice = NULL;

          // Obtiene el identificador del dispositivo seleccionado

          indice = SendMessage(hlistbox, LB_GETCURSEL, 0, 0);

          lpDevice = (LPGUID) SendMessage(hlistbox, LB_GETITEMDATA, indice, 0);

          // Creamos el objeto DirectDraw

          if (FAILED(DirectDrawCreate(lpDevice, &lpDD, NULL)))
          {
            MessageBox(NULL, "No se pudo crear el objeto DDraw", "ERROR", MB_OK);

            return 0;
          }

          // Obtenemos la interface correcta de DDraw

          if (FAILED(lpDD->QueryInterface(IID_IDirectDraw2, (LPVOID *) &lpDD2)))
          {
            MessageBox(NULL, "No se pudo obtener la interface DDraw", "ERROR", MB_OK);
            return false;
          }

          // Liberamos la interface vieja

          lpDD->Release();

          MessageBox(NULL, "DirectDraw inicializada correctamente", "EXITO", MB_OK);
        }
      break;
    case WM_DESTROY:
```

Código fuente del ejemplo (continuación)

```
      PostQuitMessage(0);
      break;

   default:
      return DefWindowProc(hwnd, Msg,
wParam, lParam);
   }

   return 0;
}

bool Init(HINSTANCE hInstance, int
nCmdShow)
{
   // Prepara una clase con algunas de las
caracteristicas de nuestra ventana

   WNDCLASS wndClass = {
                                   CS_HRE-
DRAW | CS_VREDRAW,

WindowProc,
                                   0, 0,
hInstance,
                                   NULL,

LoadCursor(NULL, IDC_ARROW),
                                   (HBRUSH)
GetStockObject(WHITE_BRUSH),
                                   NULL,
                                   "GAMEO-
VER"
                              };

   RegisterClass(&wndClass);

   // Crea nuestra ventana

   hwnd = CreateWindow("GAMEOVER",
                  "Game Over",
            WS_OVERLAPPEDWINDOW,
                  CW_USEDEFAULT,
            CW_USEDEFAULT,
                  320, 200,
                  NULL,
                  NULL,
                  hInstance,
                  NULL);

   if (!hwnd)
      return false;

   // Crea una listbox
   hlistbox = CreateWindow("LISTBOX",
                  NULL,
                  WS_VISIBLE | WS_CHILD
| LBS_NOTIFY | LBS_STANDARD,
                  20, 20,
                  280, 160,
                  hwnd,
                  NULL,
                  hInstance,
                  NULL);

   if (!hlistbox)
   {
      CloseWindow(hwnd);

      return false;
   }

   return true;
}

int PASCAL WinMain(HINSTANCE hInstance,
HINSTANCE hPrevInstance, LPSTR lpCmdLine,
int nCmdShow)
{
   MSG  msg;

   lpDD          = NULL;
   lpDD2         = NULL;

   // Nos aseguramos de que se ha creado la
ventana

   if (!Init(hInstance, nCmdShow))
      return false;

   // Mostramos la ventana

   ShowWindow(hwnd, SW_SHOWNOR-
MAL);
   UpdateWindow(hwnd);

   // Llama a la función de enumeración que
nos proporcionara los dispositivos presentes
en el sistema

   if (FAILED(DirectDrawEnumerate((LPDDE-
NUMCALLBACK) EnumDDrawDevice,
(LPVOID) hlistbox)))
      return false;

   // Bucle de mensajes

   while (GetMessage(&msg, NULL, 0, 0))
   {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
   }

   // Liberamos el objeto DDraw

   if (lpDD2)
   lpDD2->Release();

   return msg.wParam;
}
```

DirectDraw, craso error, debemos usar DirectDraw para inicializar las Direct3D, y en cierto modo, Direct3D depende de DirectDraw para mostrar sus operaciones.

Con la información de los dispositivos que nos proporciona DirectDrawEnumerate, inicializaremos un dispositivo gráfico. Debemos tener en cuenta la posibilidad de que el usuario tenga varios dispositivos gráficos, por lo que no debemos caer en la tentación de inicializar las DirectDraw con un valor de dispositivo nulo. Esto nos proporciona acceso al dispositivo gráfico por defecto, en algunos casos puede ser una forma rápida de inicializar el sistema, pero tengamos en cuenta, que no es raro que en una maquina encontremos varios dispositivos gráficos coexistiendo, una tarjeta de vídeo 2D con una aceleradora 3D.

Nuestro programa, utilizará una *listbox* para guardar los resultados proporcionados por DirectDrawEnumerate, y con la selección que realicemos, intentaremos crear el objeto DDraw, algo bastante simple, pero que puede servir de comienzo al lector interesado en la programación de DirectX.

Creación del proyecto

Los primeros pasos que debemos seguir para crear nuestro proyecto, serán crear una aplicación Win32 vacía, es decir sin código, no debemos caer en la tentación de que el propio compilador nos genere código, ya que recurrirá a las MFC, librerías de Microsoft que no se llevan del todo bien con DirectX.

Creamos posteriormente un fichero *cpp* que llamaremos *main*, el cual será nuestro fichero principal. Debemos asegurarnos que nuestro compilador tiene entre sus directorios de trabajo los que contiene el SDK de DirectX.

Includes e inicialización

Para nuestro ejemplo necesitaremos varias librerías estándar de C++, así como la librería de DirectDraw llamada *ddraw.h*. Otro parámetro indispensable será un *define*, necesario para poder utilizar la función QueryInterface, esta función nos permite obtener versiones de la interfaz DirectX diferentes a la que creamos por defecto. Podríamos pensar que Microsoft debería haber actualizado la interfaz, y que al crear un objeto DirectDraw obtuviéramos la ultima versión, pero esto eliminaría la compatibilidad hacia atrás de la que siempre hace gala Microsoft. En cualquier caso, nos basta colocar *#define INITGUID* al principio de nuestro código, para que podamos obtener la última interfaz.

Algo que no podemos olvidar es que, además de haber seleccionado los directorios de trabajo del SDK entre las opciones de nuestro compilador, el proyecto debe tener entre las librerías a linkar las propias de DirectX, por lo que debemos incluirlas entre las ya presentes.

Armando Vélez

a, utilizará
uardar los
cionados por
rate, y con la
cemos,
r el objeto
nte simple,
rvir de
interesado
n de DirectX.

proyecto
s que debe-
ear nuestro
ear una apli-
a, es decir
emos caer
que el pro-
s genere
urrirá a las
Microsoft
el todo bien

mente un
naremos
uestro
ebemos
uestro com-
sus directo-
ue contie-
X.

lo necesi-
rías están-
no la libre-
amada
metro
n *define*,
er utilizar
erface,
rmite
e la inter-
es a la que
o.
ue
aber
az, y que
irectDraw
ma ver-
naría la
atrás de
gala
ier caso,
efine
de nues-
poda-
erfaz.
s olvidar
r selec-
e trabajo
es de
royecto
rías a
ctX, por
s entre

ndo Vélez

Creando mapas 3D con Modo 8

Hace dos números, en esta misma sección explicábamos el funcionamiento del famoso modo 7 de Div, casi cuando aparecía la segunda versión de este producto. Éste lo vamos a dedicar al modo 8 y cómo crear mapas 3D con Div 2.

Vamos a dedicar la sección de este número y posteriores a explicar detalladamente cómo crear mapas 3D con *DIV 2,* usarlos en un programa y, sobre todo, cómo valerse de algunas estratagemas para conseguir superar las limitaciones del *modo8.*

El editor de mapas 3D

Todos aquellos que hayan sido, o sean, adictos al *Doom 2* y a crear niveles para él, encontrarán varias similitudes entre el editor 3D de *DIV* y editores para *Doom* como *Deep*, con pocas diferencias, salvo que en *DIV 2* no estamos usando un *engine* ya creado, puesto que eso será tarea posterior del programador.

Para crear nuestro mapa 3D usaremos la librería de texturas que viene con *DIV 2* para no complicarnos la vida. Una vez abierto el editor, seleccionando "NUEVO" en el menú Mapas 3D y haciendo doble clic en el mapa, nos encontraremos con varios campos para rellenar, botones y una gran ventana que es donde se diseña el mapa.

A la derecha hay tres casillas, que de arriba a abajo son: textura de pared, textura de techo y textura de suelo. Debajo de la textura de techo y suelo hay dos campos numéricos para rellenar, donde se indican las alturas del techo y suelo respectivamente. Tras introducir en cada una el valor de la altura y pulsar intro, una barra a la derecha nos mostrará en blanco la situación del sector con respecto a los valores mínimos y máximos de altura. Éstos son 0 y 4095, entre los que podrán oscilar las alturas de techo y suelo. Si nos fijamos, al lado de la casilla de la textura de pared y techo hay dos barras de desplazamiento. Éstas sirven para ajustar el nivel de luz de las texturas de pared y de techo y suelo. Las de techo y suelo siempre se ajustan a la vez, sólo ajustando la que hay al lado de la textura de techo.



El sector creado, con las texturas correspondientes.

A la izquierda, debajo de la ventana principal, hay varios botones con su descripción al lado. El primero es "EDICIÓN". Cuando su casilla esté seleccionada, podremos dibujar líneas que serán las paredes de un sector (que deberemos cerrar siempre, terminando en el punto de comienzo). Debajo aparecen las coordenadas X e Y del mapa en el sitio donde se encuentra el puntero del ratón. Son útiles para varias cosas que veremos más adelante.

A la derecha de "EDICIÓN" tenemos "VÉRTICE", "LÍNEA" y "SECTOR". Éstos sirven para poder seleccionar lo que sus nombres indican y modificarlos. Por ejemplo, si pulsamos "VÉRTICE" podremos elegir vértices del mapa para moverlos. En "LÍNEA", podremos elegir líneas de paredes para modificar sus texturas. En "SECTORES", las texturas de suelo y techo y las alturas. A la derecha pone "BANDERA" y un indicador numérico debajo. Teniendo "BANDERA" seleccionado, podemos insertar una bandera con el número que tengamos debajo, teniendo en cuenta que si se inserta una ya existente, la antigua se moverá a la posición de la nueva. Es decir: si hay una bandera 0 y se inserta otra como 0, la 0 antigua pasará a estar en el lugar de la nueva. Estas banderas son útiles para marcar sitios donde un proceso podrá "teletransportarse" inmediatamente con el comando *go_to_flag(<número de bandera>).*



El mapa 3D al completo, con sus tres sectores.

Inmediatamente después de bandera pone "GRID" y "SNAP", y una casilla a la derecha. "GRID" activa la rejilla, una ayuda para dibujar, y "SNAP" lo que hace es que cada vez que movamos/insertemos un vértice se ajuste a uno de los puntos de la rejilla. Estas dos opciones son extremadamente útiles, y casi obligatorias a la hora de unir dos sectores (*adjoin* como se diría en términos de *Doom*) ya que los, al menos, dos vértices que los unen deben coincidir. La casilla de la derecha es el mapa de fondo que se verá siempre que la cámara se encuentre con paredes/techos/suelos con textura 0, transparente. Y por último, a la derecha del todo, abajo, hay una pequeña casilla donde se indica el archivo FPG de las texturas.

Comenzaremos eligiendo el archivo FPG. Haciendo clic en la casilla de abajo a la derecha, se nos abrirá el "explorador" de *DIV 2*, donde nos meteremos en el direc-

con DIV 2

torio de éste, después en el directorio "WLD" y elegiremos "WLD_VIEW.FPG", con lo que tendremos todas las texturas de este FPG disponibles.

Bien, ahora vamos a empezar a crear un mapa 3D sencillo. Comenzaremos con un sector principal. Antes de ponernos a editar, conviene definir las texturas de las paredes, suelo y techo, y las alturas, para no tener que ponerlas después.

Elegiremos, por ejemplo, para las paredes, la textura 1, para el techo la 0 (para simular un exterior) y para el suelo la 9. Las alturas de techo y suelo deberían bastar con 2000 y 1500 (nunca empecéis con altura de suelo 0 a menos que estéis seguros de que ése es el sector más bajo). Comprobaremos en la barra de la derecha que, efectivamente, el área blanca se adapta a los valores de nuestro sector (recordar pulsar intro tras introducir el valor). Sería bueno que eligiéramos ya la textura de fondo. La 47 parece ser la que mejor se acopla. Ahora pulsamos F1 un par de veces para hacer un "zoom in" (el F2 sirve para lo contrario). Y vamos a dibujar un cuadrado, que es lo más simple, de 1024 de lado (usando las coordenadas X e Y y la "DISTANCIA" nos resultará más fácil determinar las medidas).

Ya tenemos un sector de comienzo (si no es cuadrado perfecto, usar "VÉRTICES" para arrastrar los vértices descolocados a su posición correcta). Ahora vamos a colocar la bandera 0 en el centro del cuadrado. En cualquier momento podemos pulsar las teclas de control para desplazarnos por el mapa. Si ahora salváramos el mapa y usáramos el programa *WLD_VIEW* (cambiando los parámetros) para verlo en 3D, sería una habitación sin techo (con el cielo de fondo), pero encerrada en cuatro paredes.

Si en algún momento un sector nos saliera realmente mal, pulsando en "SECTOR" podremos elegir el sector "defectuoso" y pulsar la tecla "SUPRIMIR" para eliminarlo.

Ahora vamos a meter un sector dentro de otro sector. Pulsamos "F2" una vez para ver nuestro mapa un poco más de lejos. Vamos a definir los valores de la altura; esta vez pondremos 2000 y 1450. A la textura del suelo y techo le bajaremos un poco la luz para que se note bien el escalón, al estar diferenciados los sectores un poco en sus texturas. Creamos un hexágono alrededor del sector inicial (no tiene que quedar perfecto). Y lo que tendríamos ahora es un sector (primero) dentro de otro sector (segundo), el primero sobresaliendo por encima del segundo.

Ahora anexionaremos un sector a otro sector. Lo normal suele ser "conectarlo" por dos vértices, pero por supuesto pueden ser más. ¿Qué hay que hacer? Elegir las texturas del nuevo sector, por ejemplo estas: 5, 11 y 12. Como podéis ver, va a ser un sector cerrado. Pero hay un problema: si anexionamos un sector con textura de techo a un sector sin textura de techo con la misma altura o menor, se notará la falta de un segundo *layer* por encima del sector con techo, y queda un "error visual" que se nota demasiado, por lo que hay que corregirlo. ¿Cómo? Poniendo el techo del sector nuevo a una altura menor. Usemos estas alturas: 1800 y 1400. Ahora nos insertamos el primer vértice EN EL MISMO SITIO donde estaba el vértice superior izquierdo del hexágono, después el segundo en el vértice superior derecho, y concluimos el sector haciendo una figura más o menos como en la figura 3. Procuraremos siempre que no queden ángulos de menos de 90º en los picos de las paredes/uniones de sectores para que no se produzca otro efecto visual que empeora la calidad del mapa.

Ahora viene un punto importante en la unión de sectores. Un sector es más bajo que otro, así que hay que definir la pared que queda por encima de la "puerta" para acceder al último sector, y también la pared que hay en el "escalón" para subir del sector último al segundo. Cuando pulsas en modo "LÍNEA" sobre una pared anexionada, se elige ésta y se iluminan los vértices del sector al que pertenece. Pulsaremos en la línea que une los dos sectores hasta que se ilumine el sector segundo. Ahora elegiremos la textura que va a tener esa primera pared "por encima de la puerta" y la del "escalón". La textura 34 servirá (para que se note la diferencia). Hay otro escalón en el primer sector pero se quedó con la textura que le asignamos a la pared primeramente (y que es también la de las paredes del sector segundo).

¡Ya podemos salvar el mapa!.

Elegiremos "GUARDAR COMO" en el menú mapas 3D, teniendo elegido este mapa, y lo guardaremos como "3DRED.WLD". Ahora podemos usar el programa *WLD_VIEW.PRG* que viene con *DIV 2* para ver nuestro mapa (acordándonos de cambiar el nombre del fichero WLD en la constante *ARCHIVO_WLD*).

Utilizando adecuadamente las texturas, nuestros escenarios quedarán perfectos

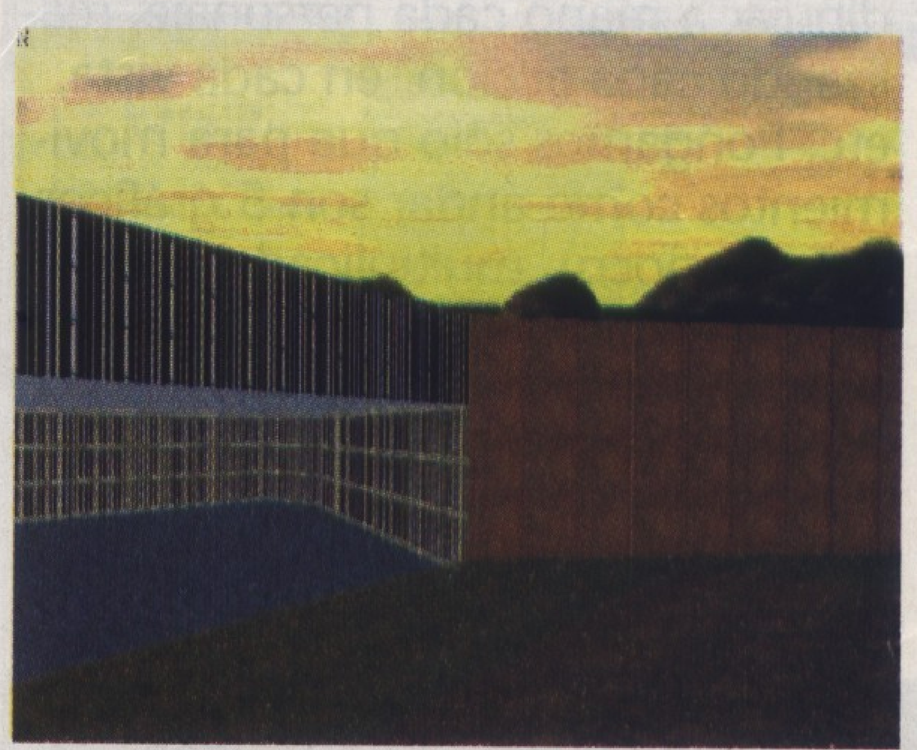
Dado que desde la programación podemos modificar las características de cada sector, si estuviera compuesta por un sólo sector sólo podríamos modificar ése. Ejemplo: si tenemos un área rectangular plana que queremos que se transforme en una escalera ascendente, tendremos que dividirla en sectores del tamaño de cada escalón, para que luego vayan aumentando su altura de suelo cada uno poco a poco, teniendo cada escalón al final una altura un poco mayor que el escalón anterior. Si este área rectangular fuera un solo sector, sería imposible "partirlo" después en escalones; tan sólo podríamos aumentar o disminuir la altura de todo el suelo a la vez.

Bueno, con esto está ya explicado el manejo del editor de mapas 3D, (una herramientas bastante potente).

Las texturas

Usar la librería de texturas del *DIV 2* es muy fácil, pero normalmente cuando hagamos un juego un poco serio querremos diseñar nuestras propias texturas. Para ello debemos pensar primero en dónde se va a desarrollar la acción, para saber, por ejemplo, si deben ser texturas naturales o texturas de edificios o interiores. Otro factor es el cuándo. No es lo mismo el suelo, techo y paredes de un castillo medieval que los de una base militar del 2040, obviamente. Para diseñar las texturas os recomiendo un buen programa de diseño 2D como *Adobe Photoshop*.

A la hora de diseñar las texturas de la pared/suelo/techo debemos tener en cuenta un par de factores. El primero es el tamaño del mapa, que debe ser un cuadrado de lado



El mapa 3D, el fondo genera una sensación mayor de tridimensionalidad.

potencia de 2. Es decir, 2x2, 4x4, 8x8, 16x16... 128x128 (el más frecuente)... El segundo factor, muy importante y a veces difícil de conseguir, es que la textura sea perfectamente cíclica. Es decir, que se pueda repetir (o *tilear*) sin que en los bordes de cada cuadrado *tileado* se note que no encaja con el borde del cuadrado de al lado. Para conseguir que una textura sea cíclica se pueden usar varios métodos. El más simple, y que quita los problemas de golpe, es ponerle bordes dibujados a la textura, es decir, que toda la textura sea una "loseta" que se repite por la pared o suelo sin que quede mal. Los procesos más complejos implican realizar una serie de *tileados*, adaptando cada vez uno de los bordes de la textura. Este proceso lleva más tiempo pero consigue un acabado más profesional. Luego están también ciertas texturas que, por su forma, son de por sí cíclicas. Y también algunas texturas que no necesitan ser cíclicas, por ejemplo, una textura que es un dibujo de un interruptor, para poner en una pared. Para conseguir que encaje la textura sin que se repita o se vea sólo una parte hay que crear un sector pequeño dentro de la pared de las medidas de la textura, y ponerlo a la altura necesaria para que encaje perfectamente.

La edición de mapas 3D con DIV 2 es realmente sencilla

Otra textura importante es la del fondo. En mapas de interiores quizá no haga falta, pero en casi todos los mapas siempre queda bien que haya alguna ventana o habitación sin techo a través de la cual se pueda ver el fondo, ya que con el movimiento se genera una sensación mayor de 3D.

Los personajes

Otro de los factores importantes en todo juego 3D es el diseño de los personajes. Los personajes necesariamente deberemos diseñarlos con un programa de diseño 3D como *3D Studio Max*, dado que resulta demasiado engorroso tener que dibujar a mano cada personaje, realizando cada acción, en cada vista, etc. Pongamos sólo que para movimientos como andar son 5 gráficos (que es poco), multiplicado por 8 vistas (lo mínimo para que quede bien) ya son ¡¡40 gráficos!! Sumándole todos los demás movimientos, y multiplicando por el número de personajes (no van a ser todos iguales, enemigos y personaje principal) obtenemos cifras que pueden llegar a rondar los 500 gráficos (en cualquier juego mínimamente decente con distintos tipos de enemigos) y muchas veces más, por lo que no hay que poner en duda que el diseño de los personajes puede llegar a ser MUY pesado.

Para diseñar los personajes podemos usar también el generador de *sprites* de *DIV 2*, aunque su único defecto es que está bastante limitado en cuanto a movimientos se refiere, dado que no podemos añadir nuevos, ni editar los existentes. Además, uno se termina aburriendo de los únicos tres modelos (hombre, mujer y niño) que hay. Por lo que lo mejor es usar un programa externo, que si bien suelen ser más pesados (salvar gráfico a gráfico, cada vista renderizada, girar el personaje...) el modelo 3D se mueve totalmente a gusto del diseñador, e incluso si uno es un manitas del diseño, puede que quede todavía mejor que con el generador de *sprites*. Es cuestión de práctica.

Normalmente, en los juegos en 3D la cámara está en 1ª persona, tan sólo hace falta diseñar las armas vistas desde el punto de vista del protagonista, para que aparezcan en la mano del personaje al seleccionarlas. Y claro, en juegos con cámara en 3ª persona (desde fuera) o modo multijugador, se necesita el diseño del actor del juego.

Los objetos

Para disminuir los requerimientos de memoria/espacio del juego (y facilitar la tarea al programador) normalmente los objetos son procesos que usan *sprites* fijos. ¿Qué son los *sprites* fijos? Son dibujos en 2D que, los mires por donde los mires en el mapa 3D, siempre muestran el mismo lado, esto es, tienen un sólo gráfico de vista. Puede que con algunos tipos de objetos esto quede algo irreal, pero bueno, es algo que ocurre en todos los juegos 3D basados en *sprites*. Normalmente, sólo los objetos inertes e ítems que puede recoger el personaje usan *sprites* fijos. Si alguien quiere hacer un juego con *DIV* para Pentiums III con 128 Mb de RAM puede perfectamente, si quiere, ponerle hasta 16 vistas distintas a un objeto (y más), pero son ganas de complicarse la vida.

Otros elementos que usaremos con *sprites* fijos serán, por ejemplo, las balas que sean visibles (como lasers), los impactos de balas en pared/enemigos, las explosiones... todo lo que no requiera que el personaje sepa hacia dónde está "mirando".

Últimos consejos

Os recomiendo que nunca pongáis la misma luz en paredes adyacentes ni en sectores contiguos. Tampoco en los suelos, aunque sólo varíe en un punto la luz. Podéis comprobar que si todas las texturas están a la misma luz a veces no se distingue bien donde hay un escalón, y las paredes quedan clónicas. Variando la luz un poco esto se evita.



Un personaje diseñado con 3D Studio para hallar las vistas.

A la hora de copiar sectores (función no incluida, por desgracia, en *DIV 2*) lo mejor es que os fijéis en las coordenadas X e Y de cada vértice del sector a copiar, para saber la distancia de los vectores que separan cada vértice y su dirección, cosa que facilitará mucho el hacer una réplica manual del sector original.

Cuando diseñéis los personajes o *sprites* fijos para cualquier objeto/efecto debéis fijaros en dónde ponéis el punto de control 0, puesto que luego cuando defináis la *height* de su proceso debéis tener en cuenta que es la distancia que añadirá desde la Z en la que se encuentra (altura) al punto 0. Así que si su *height* es muy baja y la z está a ras de suelo, con el punto 0 en la parte de arriba del gráfico, nos quedará un enanito (no en tamaño sino en altura, sin quedar deformado).

En el próximo número y explicaremos cómo valernos de diversas estratagemas para conseguir olvidarnos de la limitación de un solo *layer*, y otras barreras que con imaginación podremos romper. Hasta la próxima.

Si tenéis cualquier idea, sugerencia, duda, etc. mandádmela a mi e-m@il para que en próximos números pueda intentar aclararla y todos aprendamos más.

Fermín Vicente

Lcc-Win32

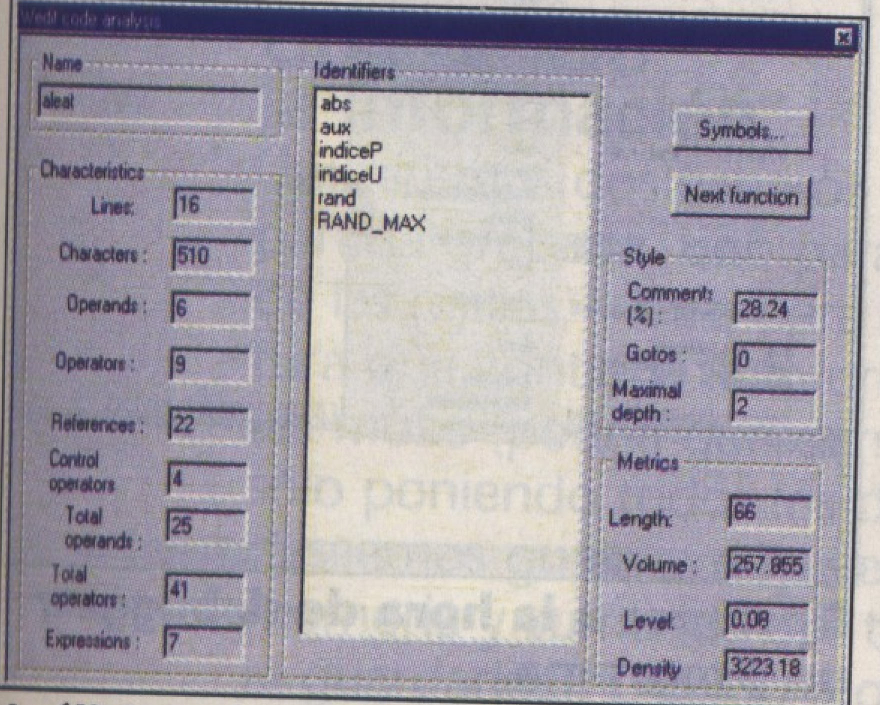
La herramienta definitiva de programación en C

En este número hablaremos del paquete Lcc – Win32, herramienta imprescindible para la programación en Windows 95. En este paquete viene el editor de textos Wedit, que junto con el programa de compilación de C, Lcc, hace un gran equipo para disfrute de los programadores.

A los programadores de C lo que siempre nos ha hecho sudar chorros de tinta es la cantidad de tareas repetitivas y documentación que debe acompañar a cada programa que hacemos. Siempre nos hemos visto obligados a hacerlo a mano, sin otra ayuda que el típico entorno gráfico del editor más común, que nos deja todo el trabajo a nosotros. No obstante, teniendo en mente este problema tan común, Fraser, Hanson y Navia se decidieron por fin a hacer un entorno de programación orientado "realmente" a programadores: el *Lcc – Win32*. Y lo mejor de todo: el paquete es totalmente *freeware*, aunque los autores agradecen que, si ha sido de alguna utilidad el programa y se puede, se haga una donación simbólica para mantener vivo el programa.

Instalación y configuración

La instalación del programa es muy sencilla; éste viene en un paquete de distribución autoextraible, donde lo único que hay que indicar es el directorio de la instalación. Una vez instalado se crean varios accesos directos, uno en el escritorio y otro en el menú inicio, que ejecutan el programa.



Análisis exhaustivo de funciones.

El entorno *Wedit*, además de utilizar el compilador *Lcc* para la compilación en C, tiene las opciones de compilar los distintos programas en lenguajes *Ensamblador*, *Pascal* y *Fortran*.

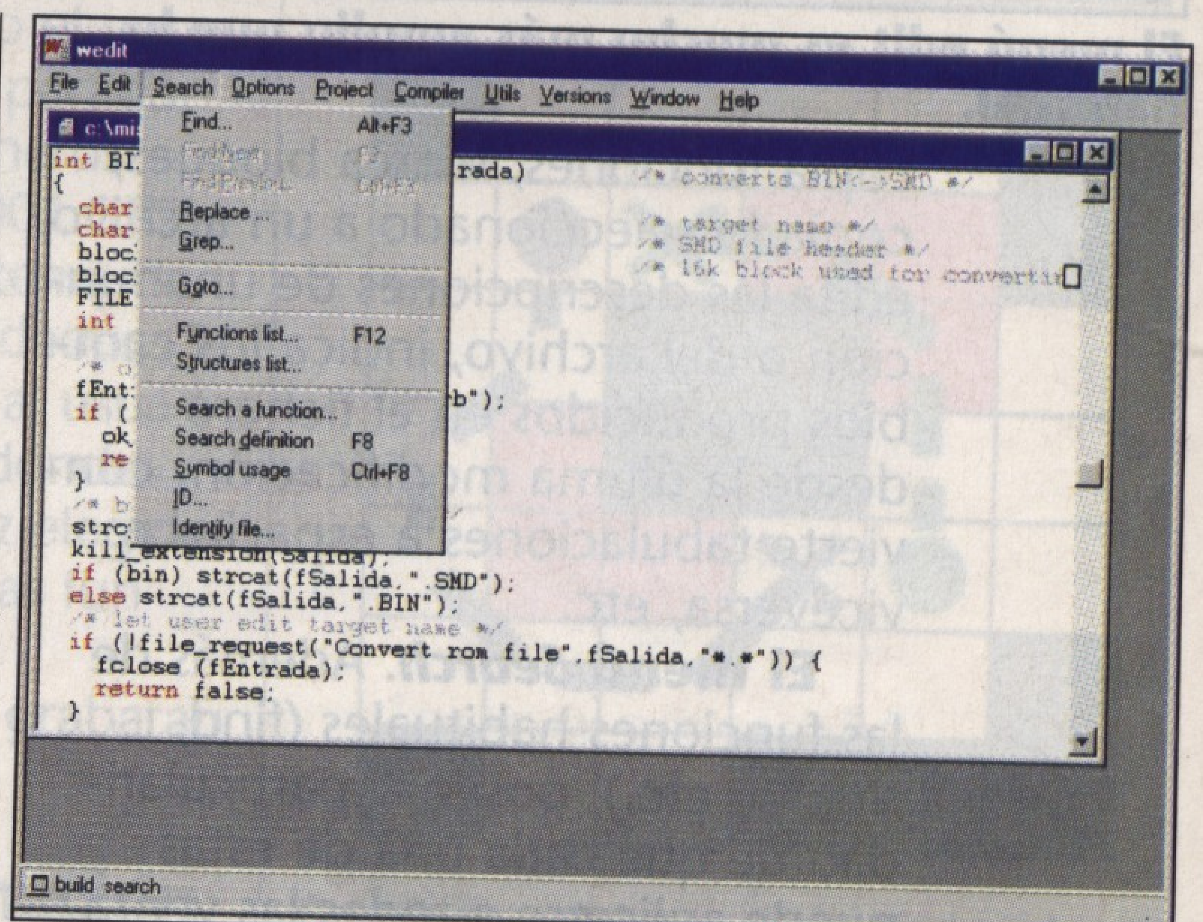
El programa posee todas las opciones que puede tener cualquier programa de *Windows*, ser una aplicación de consola, de *Windows*, una DLL (*Dynamic Link Library*) o una librería estática. Con todo esto, y las herramientas que comentaremos más adelante, este programa se alza sobre casi todos sus competidores en este campo.

Un breve repaso

Abrir un fichero. Para empezar, es necesario crear o abrir un fichero, éstos son los tipos que soporta *Lcc*:

.c: archivos de código C.
.h: archivos cabecera de código C.
.f: archivos de *Fortran*.
.asm: archivos de ensamblador.
.hpp: archivos cabecera de C++.
.pas: archivos de *Pascal*.
.exe: archivos ejecutables (posee un editor interno para la edición de archivos binarios).

Abrir un proyecto. Hay casos en que es necesario compilar varios ficheros y unirlos (*linkarlos*) conjuntamente. Para hacer esto, se define lo que se llama un proyecto, donde se mantendrá el número de ficheros a incluir y cuáles son. Tanto para crear un proyecto, como para borrarlo y modificarlo tendremos que ir al menú *Project*. Al crearlo, nos preguntará nuestro nombre, el del proyecto, el directorio de trabajo y el tipo de aplicación que deseamos crear. Una vez indicado todo esto, podremos elegir entre el *Application Wizard* o crear nosotros mismos el esqueleto del programa.



Las opciones de búsqueda son numerosas.

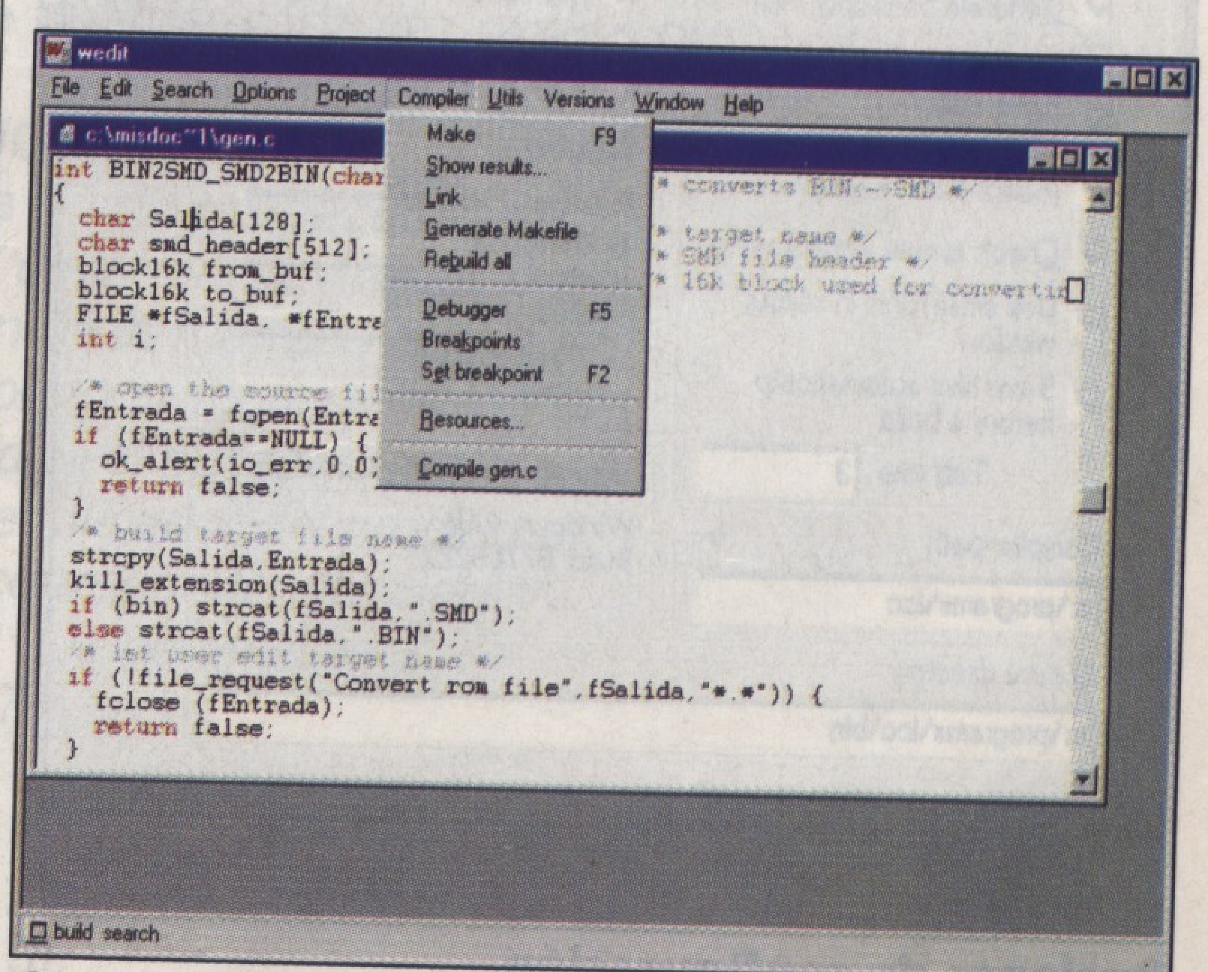
El *Application Wizard* nos preguntará que tipo de programa será y que librerías utilizará. Una vez hecho esto se creará un fichero con el esqueleto del programa.

En este punto deberemos introducir las opciones de compilación, que por cierto son bastantes. Hay que estar familiarizado con ellas, aunque, para programadores noveles, es mejor dejar los valores que vienen por defecto.

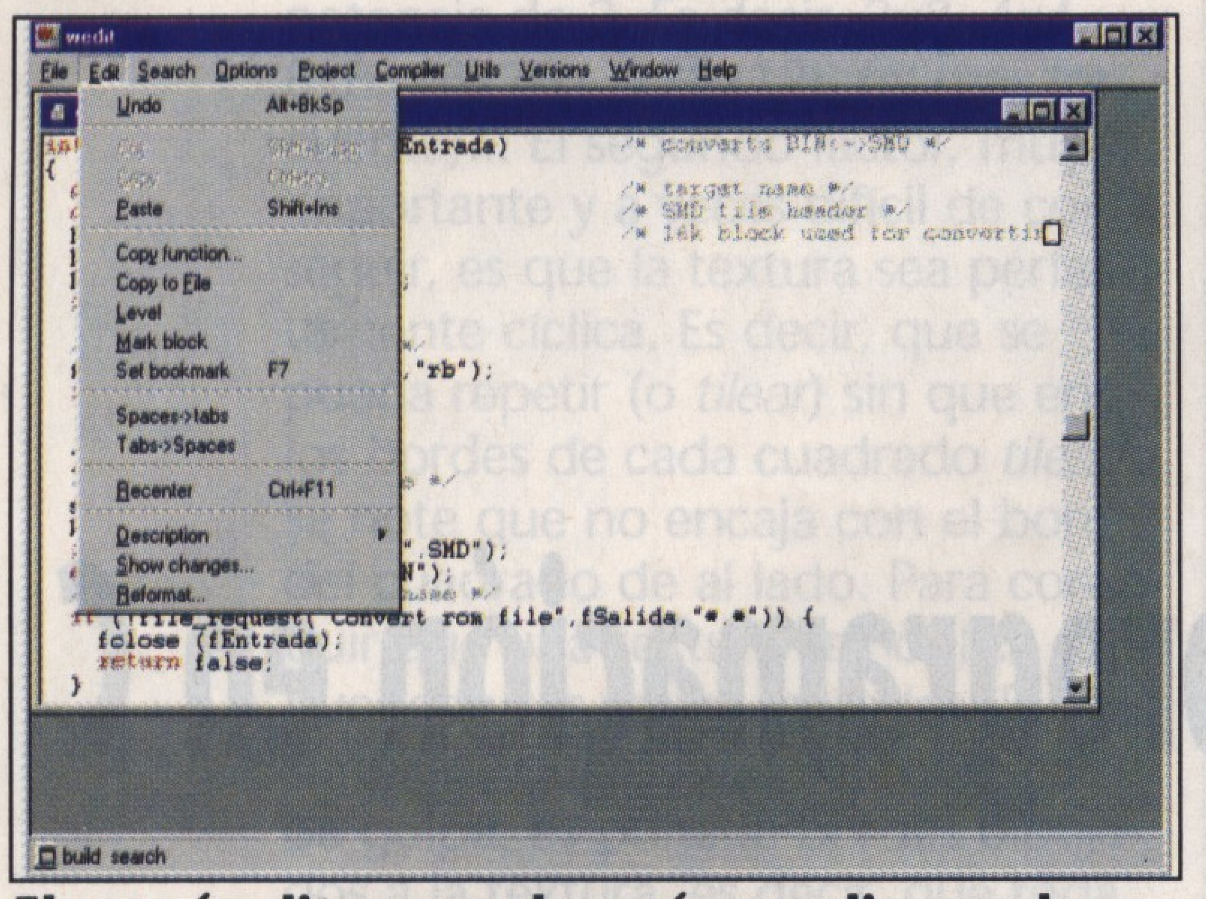
> Nunca en la vida había sido tan fácil la programación en C

Una vez hecho todo esto, ya estamos totalmente preparados para empezar a programar.

El menú *Edit*. Nunca había visto un menú más completito,



Aspecto del menú compiler.



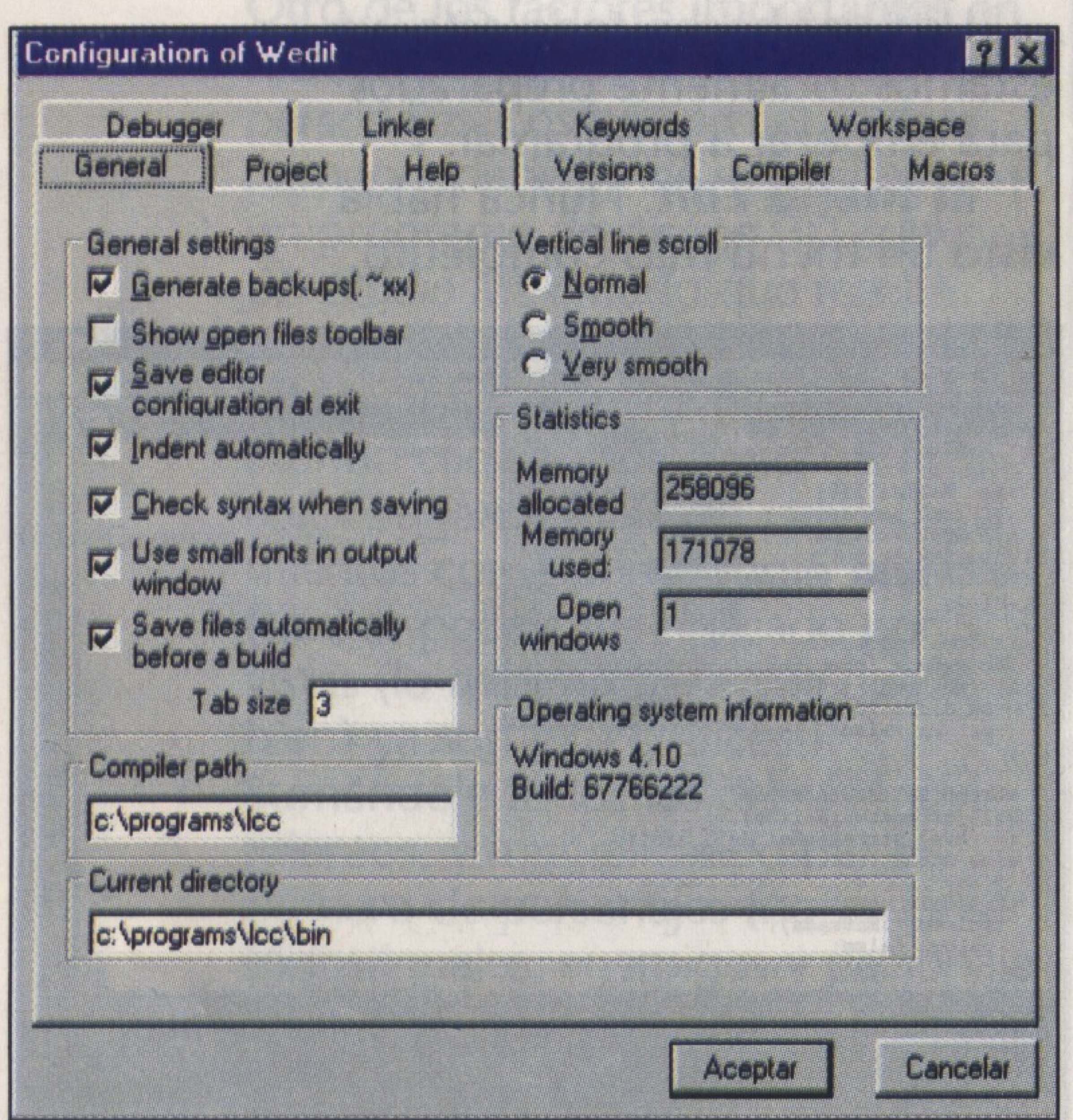
El menú edit es mucho más amplio que lo habitual.

copia funciones, marca bloques, copia lo seleccionado a un archivo, edita las descripciones de una función o del archivo, indica los cambios producidos en el fichero desde la última modificación, convierte tabulaciones a espacios o viceversa, etc.

El menú *Search*. Además de las funciones habituales (find, search, etc.), posee la particularidad de que cada una de éstas puede aplicarse a todas las ventanas abiertas. También posee una utilidad similar al *grep* de *Unix*. Puedes moverte por los códigos utilizando *bookmarks*, puedes buscar una función o sacar la lista de todas las funciones o estructuras del programa, en resumen, es bastante funcional.

Con esta utilidad será mucho más fácil documentar cualquier tipo de programa

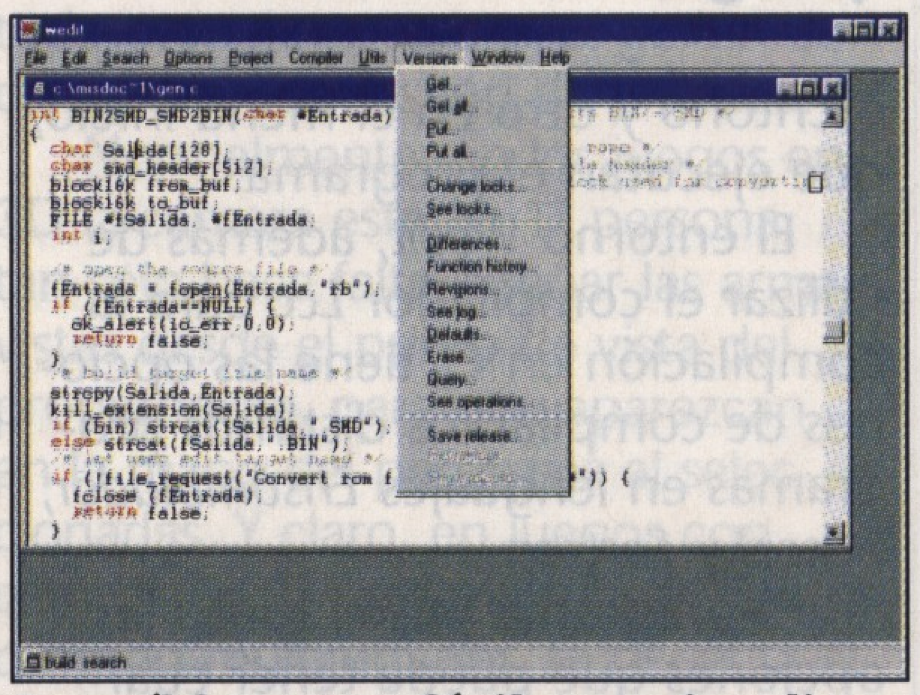
El menú *Options*. Aquí es donde se pueden configurar muchas de las opciones del programa, por poner un ejemplo, si estamos acostumbrados a las teclas del *Borland TurboC* podríamos cambiar la tecla de *debuggear* el programa de F5 a F7, y asignar más teclas a más opciones que, a lo mejor, no tienen asignada ninguna. También se pueden asociar programas que funcionarán como utilidades externas.



Opciones de configuración.

El menú de configuración es muy extenso, y va desde las opciones más básicas hasta la creación de macros, pasando por todas las opciones de distintos grados de optimización de código a la hora de *linkar* programas, o la personalización de los colores de pantalla, todo un lujo.

El menú *Project*. En este menú crearemos proyectos, y abriremos, cerraremos o borraremos proyectos previamente definidos. Para la creación de proyectos, el *Wedit* crea una llave en el registro de *Windows*, donde almacena todos los datos del mismo.



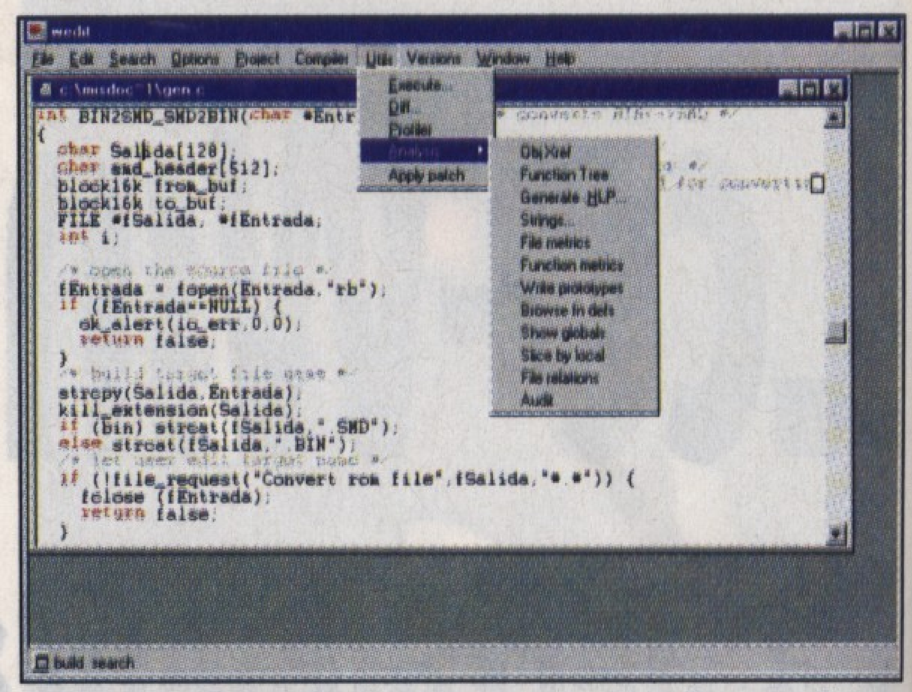
Increíblemente fácil manejar distintas versiones.

El menú *Compiler*. En este menú encontraremos las opciones más comunes de los compiladores: *compile*, *make*, *link*, *rebuild all*, *debugger*, *set breakpoint*, etc.

El menú *Debug*. Si marcamos la opción *debugger*, se sustituirá el menú *Compiler* por el menú *Debug*, en el cual podremos encontrar todas las herramientas que existen: avance en el programa de distintas maneras (*execute*, *step in*, *step out*, *run to cursor*, *stop debugging*, etc.), visualización de distintos datos (*watches*, pila, memoria, registros de la CPU, etc.) y organización de *breakpoints*.

El menú *Utils*. Esta es la joya de la corona del programa. Existen gran variedad de utilidades para el análisis de proyectos: análisis de referencias a objetos, árboles de funciones, generación de archivos de ayuda, análisis métrico de funciones y ficheros, escritura de los prototipos en un fichero, relaciones entre ficheros, y más herramientas que realizan un análisis exhaustivo de las propiedades de un proyecto.

El menú *Versions*. Se ha incluido en el *Wedit* una utilidad para la creación y manejo de distintas versiones del mismo proyecto. Lleva una base de datos con las distintas versiones hasta el momento, con comentarios, fechas de lanzamiento, cambios entre distintas versiones, modificaciones por hacer, historia de las funciones, revisiones, y un sinfín de utilidades para llevar ordenadamente el lanzamiento de una versión de un programa.



El menú más interesante de todos.

WeditRes es un pequeño programa para la creación del esqueleto de la disposición de ventanas (*resources*) en pantalla. Es totalmente gráfico, colocando los botones, listas, listas de opciones o cajas seleccionables en el lugar de la ventana que prefiramos, facilitándonos así la codificación de esta parte del programa, que es totalmente automática.

Para terminar

Si queréis más información sobre *Lcc-Win32*, los datos de *Jacob Navia*, creador del programa, son:

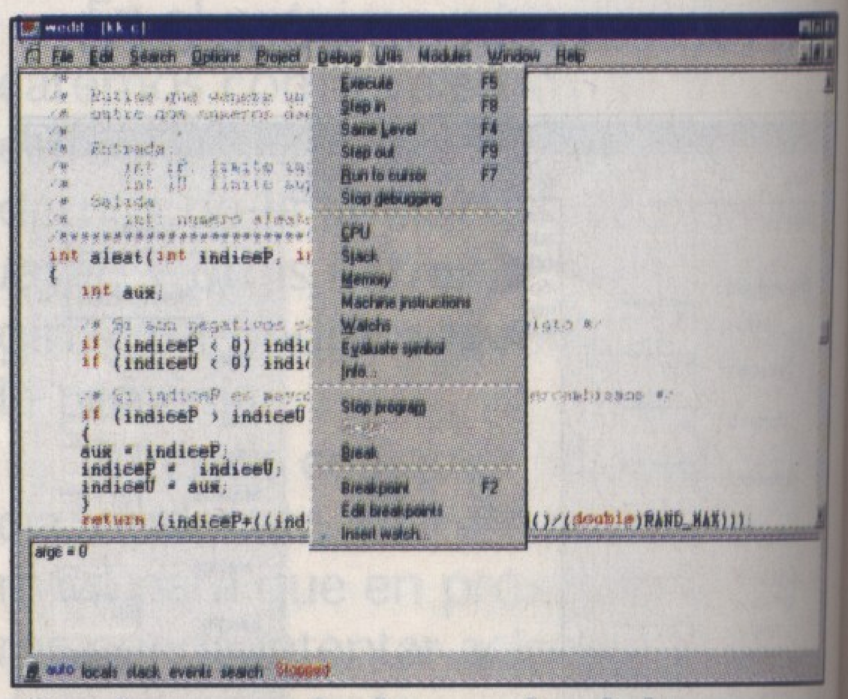
Jacob Navia.
Logiciels/Informatique.
41 rue Maurice Ravel.
93430 Villetaneuse.
France.

***Lcc* mailing list**:
majordomo@cs.princeton.edu (poniendo en el cuerpo del mensaje: **subscribe lcc**).

Web:
http://www.cs.virginia.edu/~lcc-win32

Hasta el próximo número de la revista.

Javier Fernández



Opciones a la hora de debuggear un programa.

Seguimos aprendiendo

Programación de Juegos de Estrategia - 4

Como dijimos en el pasado artículo, hoy veremos cómo hacer que los mobs se muevan por el mapa sin quedarse dando vueltas alrededor de un obstáculo. Veremos también cómo guardar y cargar partidas (imprescindible en un juego de estrategia, ya que las partidas suelen alargarse varios días).

Lo primero que veremos hoy será una forma de guardar el estado de la partida en un momento dado, y para esto debemos ver qué datos describen el estado de la partida.

Uno de los datos más importantes es el estado actual del mapa, que puede haber sido modificado talando arboles, reventando montañas, etc. Otro dato es la posición de los mobs, destino, vida, etc. Si miramos atrás, veremos que todos los datos de los mobs están en una estructura global, lo que nos facilita el grabado de esta información, ya que si tuviéramos que mirar proceso a proceso sus locales no podríamos guardarlas en un solo fichero sino en tantos como procesos hubiera. Con esta estructura podemos saber el número de mobs y todos sus datos.

Los datos podrían guardarse en un solo fichero poniendo todos los datos que deseemos guardar adyacentes en memoria y sumando sus tamaños al guardarlos

Guardando la información.

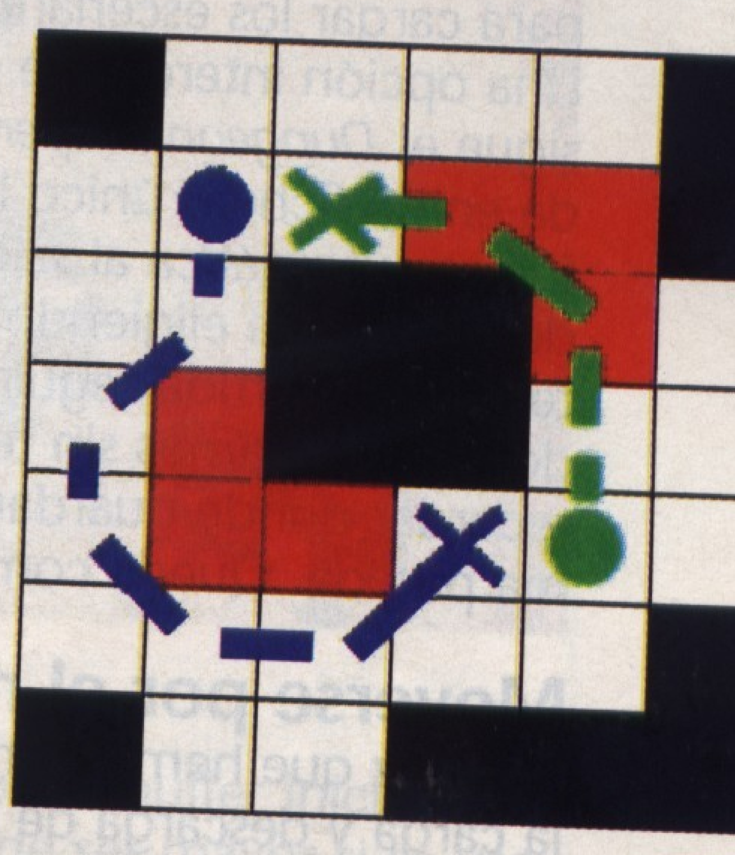
Utilizaremos dos ficheros para cada partida grabada, uno para los datos de las rejillas, y otro para la estructura que contiene la información de los mobs (podría guardarse en uno sólo poniendo todos los datos que deseemos guardar adyacentes en memoria y sumando sus tamaños al guardarlos). Debido a que DIV 1 no permite manejo de strings, no podemos pedirle al usuario el nombre con el cual guardar la partida, ni generar nombres de ficheros genéricos (Ejemplo: SAVE001.$$$), por lo que usaremos el sistema de tener varios nombres guardados (5 en nuestro caso) y pedirle al usuario que elija una posición donde guardar la partida. Una vez elegida la posición, llamaremos a las funciones Guardar_Mapa y Guardar_Bichos, las cuales grabaran en su sitio los datos.

Cargando la información

Cargar la información se realiza de forma similar a guardarla, pero además, se debe calcular toda la información que se obtiene a partir de estos datos. Lo mas sencillo es calcular el número de mobs: recorremos la estructura contando el número de posiciones ocupadas. Lo siguiente es crear los procesos mob existentes en la partida cargada (tras matar los actuales).

Aquí podemos pensar que el padre de todos los mobs será la función de Cargar_Bichos. Tenemos dos opciones: dejar huérfanos a todos los mobs nada mas cargarlos (salir de la función de carga), o mantener a la función creadora viva. Mantenerla viva permite hacer un sig_kill_tree para matar a todos los mobs, mientras que si son huérfanos el sistema es algo más complicado (o se guarda cada id, o cada mob se "suicida" al poner algún flag a 1) aunque desperdicia menos recursos. Optaremos por la segunda opción y simularemos las señales mediante un *flag* que todos los mobs comprobarán y saldrán cuando sea *true*. El sistema es parecido al de la pausa. Cuando una variable global se ponga a 1, se sale del bucle. De hecho, la pausa simula un *sig_freeze* para todos menos para la función de pausa.

En los cuadros se muestran las funciones para cargar y guardar partidas. Podemos usarlas también



Caminos mínimos para llegar desde una bola a la cruz de su color.

Código de la estructura para los datos de los bichos (mobs) y las rejillas.

Esto es lo que hay que guardar cada vez que se graba una partida:

```
struct mobs[MAX_MOBS]
    ocupado; /*marca si esta ocupado o no*/
    casilla; /*posicion de la matriz que ocupa */
    destino; /*posicion de la matriz a la que va */
    tipo; /*mosca,escorpion,tanque... */
    flag_volador; /*indica si puede volar o no*/
    identificador; /*identificador del proceso al */
                   /*que corresponden los datos*/
    bando;/*bando al que pertenecen*/
    vida; /*Obvio ¨no?*/
end

rejilla_del_mapa[10*10];
rejilla_de_juego[10*10];
```

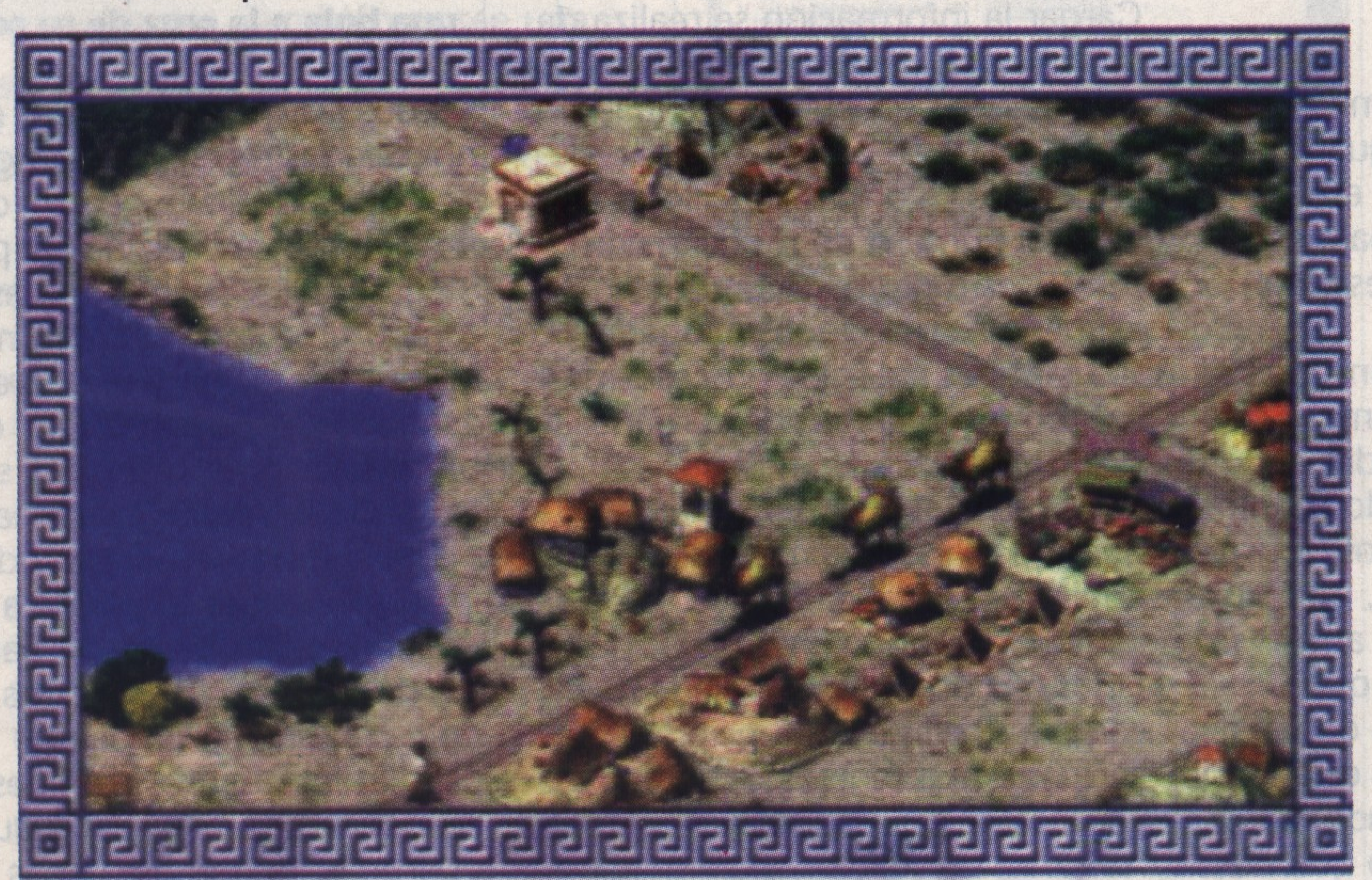
Glosario de términos

Flag: variable que se usa como indicador booleano. Si tiene valor impar (1) es verdadero, en cambio si lo tiene par (0) es falso.
Array: un grupo de datos colocados en fila en memoria. Se accede al elemento N-esimo poniendo el nombre del array y entre corchetes [] el índice del elemento al que se desea acceder.
Buffer: array que se usa como almacén temporal de datos.
Recursividad: técnica de programación en la que una función se llama a si misma pero con diferentes argumentos.
Pseudocódigo: Forma de representar el código de algoritmo de forma que cualquier programador pueda entenderlo sea cual sea el lenguaje que piensa utilizar.

para cargar los escenarios iniciales. Una opción interesante es la que sigue el *Dungeon Keeper*, que guarda en un fichero único la partida de forma automática al salir del juego, de modo que eligiendo la opción recuperar partida seguiremos donde lo dejamos sin tener que recordar donde guardamos la última partida. Queda como idea.

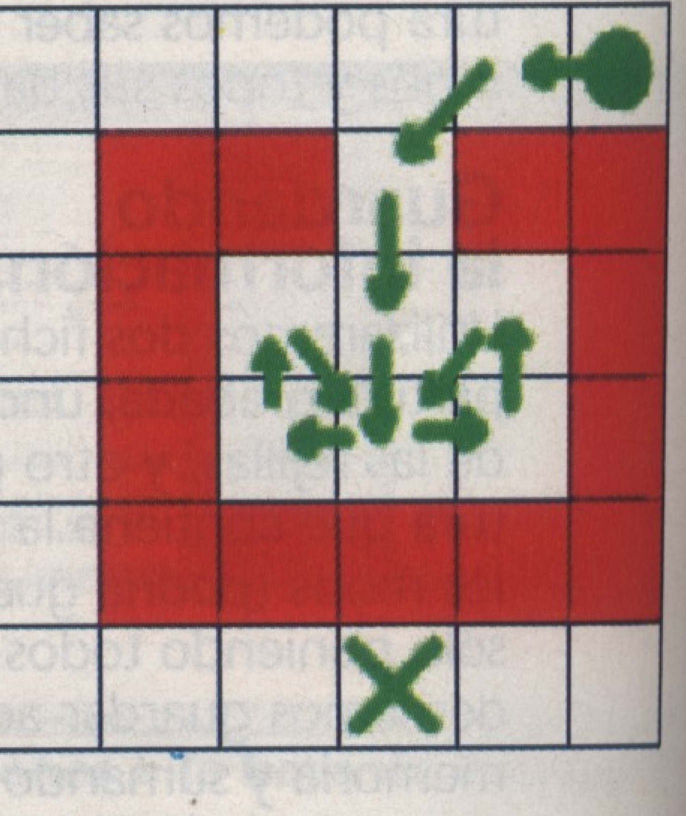
Moverse por el mapa.

Una vez que hemos terminado con la carga y descarga de la información de una partida, vamos a ver las diferentes formas de moverse por el mapa. La mas sencilla es calcular qué casilla acerca más a nuestro mob al destino que tiene fijado y avanzar hasta ella. Esto lo podemos saber calculando el ángulo que forma nuestra posición con el destino eligiendo la casilla que esté en ese ángulo (cada casilla tiene una franja de ángulos asociada). También se puede calcular la distancia al objetivo desde cada una de las casillas adyacentes, y elegir el mínimo, pero esto es demasiado lento. Hasta aquí todo es muy sencillo, pero el problema aparece cuando la casilla elegida esta ocupada (por otro mob, por montañas, etc). He aquí donde es necesario aplicar un método de búsqueda de caminos. En un artículo anterior aparecía el pseudocódigo de un algoritmo que encuentra el camino mínimo, pero tiene un problema en los juegos: es demasiado costoso para que cada mob esté ejecutándolo si tiene que ir a algún sitio. Las opciones son muchas, desde el más sencillo, como el del Warcraft, al más complejo, como el que acabamos de mencionar. Si optamos por uno sencillo, los mapeados deberán estar diseñados para que los movimientos de los mobs sean realistas (quién no ha perdido una unidad y cuando la ha encontrado estaba en un bosque dando vueltas como un idiota). El más sencillo es aquel que cuando el mob encuentra un obstáculo, opta por seguir por la siguiente casilla que más le acerca. Este sencillo algoritmo funciona cuando el mob choca con obstáculos que no tienen concavidades, es decir, mas o menos circulares. Una modificación de este algoritmo permite al mob salvar un numero determinado de casillas en un buffer (array para guardar posiciones del mapa) para evitar pasar de nuevo por ellas. Esta sencilla modificación permite al mob salir de pequeñas concavidades en las que se pueda meter (según el tamaño del buffer podrá salir de concavida-





Un problema en la búsqueda de caminos es la modificación del entorno. Lo que antes estaba libre, ahora no lo está.



Problema que existe al quedarse un mob en un obstáculo cóncavo.



Buscar caminos es diferente para una unidad aérea y para una terrestre.

des más grandes o más pequeñas). Otro algoritmo similar aunque algo más costoso es haciendo la búsqueda de forma recursiva, es decir, el mob llama a una función que se llama a si misma sobre la casilla más cercana al destino. Si ésta falla, luego se llama sobre la siguiente mas cercana, y así con las ocho casillas adyacentes. De esta forma se llega recursivamente al objetivo, y cuando llega, va devolviendo TRUE y se va guardando el camino recorrido. Se debe limitar la profundidad de recursión al tamaño del buffer, o que el mob no conozca el camino entero sino sub-objetivos, es decir, aquellas posiciones en las que debe cambiar de sentido, o solo guardar 1 de cada N posiciones y mantener otro buffer de N posiciones donde buscar caminos (de otra forma mas rápida) que unan esos sub-objetivos, etc. Con este algoritmo, seguro que llegamos al destino, pero puede que tarde un poco en calcularlo, y si andamos escasos de memoria, puede incluso cascar el juego, lo cual nunca hace quedar muy bien a un juego ¿verdad?. Para "evitar" que el juego casque debemos limitar la profundidad de recursión, para que sepamos cuanta memoria como máximo va a necesitar el juego. Con esta limitación, puede que a veces no consiga llegar nunca al objetivo, así que la nueva condición de finalización (cuando deja la función de llamarse a si misma) no será sólo cuando encuentre el objetivo o un obstáculo, sino cuando llegue al límite también. El límite no tiene porque ser el tamaño del buffer inicial, sino que puede ser mayor si se usa la opción de 1 de cada N. Por ultimo, podría implementarse el algoritmo de exploragrafos que mencionábamos antes, y cuyo pseudocódigo aparece en el número 2 de la revista en la sección del francotirador. Este algoritmo no obtiene un buen camino hacia el objetivo. Obtiene el mejor camino pero a costa de la velocidad.

Cada mob se "suicida" al poner determinado flag a 1 simulando una señal de kill

Y en el próximo número...

En el próximo número haremos un pequeño editor de fases para nuestro juego.

Espero que nos leamos pronto.

Emilio Llamas Alba

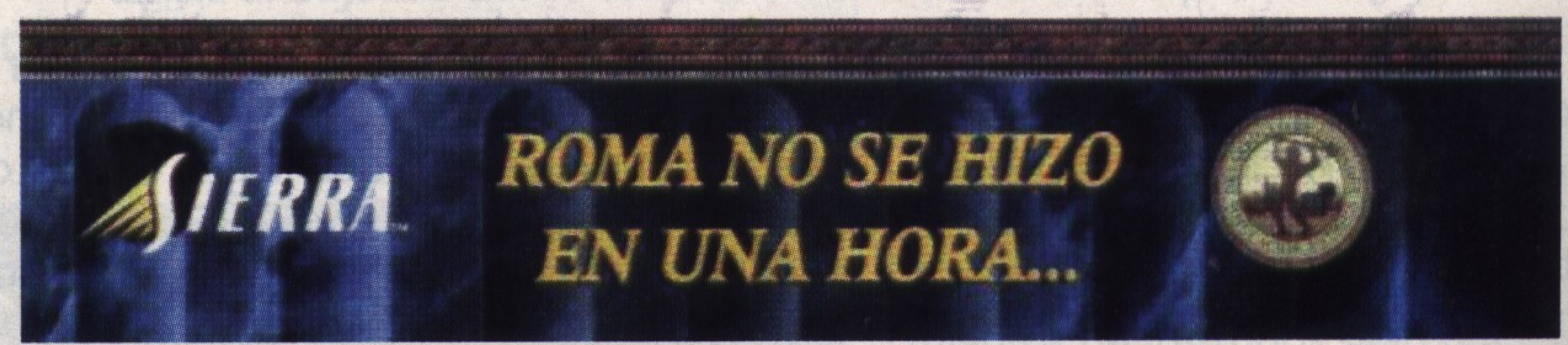
Pseudocódigo ExploraGrafos:

1) Guardar en una cola (ABIERTOS) el punto de origen (S).
2) Inicializar a vacía otra cola (CERRADOS).
3) Si ABIERTOS esta vacía, algo ha ido mal... no se pudo encontrar un camino.
4) Sacamos el primer elemento de ABIERTOS y lo guardamos en una variable N y en CERRADOS.
5) Si N es un destino, entonces calculamos el camino gracias a los apuntadores que unen N con S (el origen).
6) "Expandimos" N, es decir metemos todas las casillas adyacentes a N (y accesibles) en X, siempre y cuando estas casillas no fueran ascendientes de N (tuvieran un apuntador desde N hasta ellas).
7) Poner un apuntador a N desde los "sucesores" de N que no estén ni en ABIERTOS ni en CERRADOS. Y para todo H contenida en X:
7.1) Si H esta en ABIERTOS o en CERRADOS, comprobamos si el camino que lleva hasta H pasando por N es mejor que el que tenía marcado hasta ese momento, en cuyo caso, modificaremos el apuntador que tenia H para que apunte a N.
7.2) Si H esta en CERRADOS, decidir para los descendientes de H si el camino a través de H es mejor que el que tenían hasta ese momento, en cuyo caso, modificaremos los apuntadores de los descendientes de H.
8) Reordenar ABIERTOS de acuerdo a una heurística (primeros los que más cerca estén en línea recta al destino, y últimos los que mas lejos estén).

El Francotirador







Caesar III

El número pasado nos quedamos con las ganas de leer el francotirador. Hoy volvemos con fuerza renovada a destripar otro juego. El juego que nos ocupa mezcla la estrategia con la simulación de ciudades, y lo hace muy bien. Veamos algunas cuestiones del juego.

Empezaremos comentando la cuestión gráfica del juego.

Gráficos
El escenario del juego se representa en una perspectiva isométrica que permite ver más edificios que la perspectiva frontal elevada. Cada tipo de edificio tiene sólo una representación, pero la variedad de los mismos es apabullante. Los mobs tienen varias posiciones para hacer las animaciones de movimiento en las cinco orientaciones necesarias (tres de las ocho se hacen espejando). Las edificaciones normales están diseñadas para ocupar cuadrados de NxN casillas, para que al rotar puedan utilizarse los mismos gráficos.
Tan sólo el portón, las murallas, los acueductos y los fuertes pueden adoptar distintas posiciones según la orientación. No tiene los mejores gráficos del momento, pero están cuidados.

Inteligencia Artificial
La inteligencia artificial de este juego destaca por su variedad y sencillez, al igual que ocurría con los gráficos. No tiene la IA más avanzada, pero con la que tiene consigue hacértelo pasar mal. Los personajes civiles tan sólo se mueven errantes por las calles de la ciudad hasta que llegan de nuevo a su punto de partida, o llegan demasiado lejos, momento en el cual desaparecen sin más. Debe ser el jugador el que decida como organizar las calles. En cambio, los militares enemigos (los nuestros hacen lo que se les dice) aprenden de ataques anteriores, eligen puntos débiles en las murallas y atacan por diferentes flancos, distribuyendo las fuerzas según las defensas a las que se enfrentan. De todas formas, a veces se les puede engañar poniendo un trozo de muralla en mitad del camino, lo que entretendrá a parte de las fuerzas mientras fulminamos al resto. Parece ser que tienen como prioridad destruir las murallas independientemente de si hay o no otra forma de entrar. El algoritmo de búsqueda de caminos es el de explorágrafos (o uno similar), ya que obtiene siempre el camino mínimo. Lo que realmente dificulta el juego son la economía y la gran cantidad de parámetros que debe uno tener en cuenta para triunfar.



Caesar III es un ejemplo a seguir. El equilibrio "gráficos – IA" es perfecto.

La construcción de los diálogos



Preparando la interactividad

En esta nueva entrega del curso de programación de RPG's veremos como implementar los diálogos en un juego. Esto puede parecer una cosa muy simple, pero es necesario tratarlo a la hora de hacer un RPG ya que estos están plagados de diálogos.

El código está dividido en 2 programas. El primer programa se encargará de grabar los textos a un fichero en el disco, y el segundo se encargará de leerlos, interpretarlos y representar los textos correctamente.

• Programa 1:

```
    Program Escribe_textos;
    Private
        String
texto[5000]=”l1$l2$l3$l4$l5$l6
$l7$l8$l9$l10$l11$l12$l13$l14
$l15$16$”;
        fichero1;
        idtext;
        struct fixero
            inicio[10];
            longitud[10];
        end
    Begin

fichero1=fopen(“datos.maq”,”w”);

fixero.inicio[idtext]=ftell(fichero1);

fwrite(&texto,strlen(texto),fichero1);

fixero.longitud[idtext]=ftell(fichero1);
        texto=”
t1$t2$t3$t4$t5$t6$t7$”;
        idtext++;

Fixero.inicio[idtext]=ftell(fichero1);

fwrite(&texto,strlen(texto),fichero1);

fixero.longitud[idtext]=ftell
(fichero1)-fixero.inicio[idtext];
    fclose(fichero1);
    save(“a.maq”,offset fixero,
sizeof(fixero) );
    compress_file(“datos.maq”);

encode_file(“datos.maq”,”MAQUI”);
end
```

```
Programa 2:
Program Dialogos;
Begin
    set_mode(m320x240);
    select_text(0);
    select_text(1);
End
/*
  PARAMETROS:
  - IDTEXT -> NUMERO DEL
TEXTO QUE QUEREMOS ESCRI-
BIR
  */
Function select_text(int idtext)
Private
    fichero1;
    string a[5000];
    struct fixero
        inicio[10];
        longitud[10];
    end
```



Los diálogos o conversaciones dentro de un juego del género de rol son parte virtual del mismo.

Código ejemplo 1.

```
Program Escribe_textos;
Private
    String texto[5000]=" Esto es un$ejemplo del progrma$de
dialogos$para la revista$Divmania numero 4.$";
    fichero1;
    idtext;
     struct fixero
        inicio[10];
        longitud[10];
     end
Begin
    fichero1=fopen("datos.maq","w");
    fixero.inicio[idtext]=ftell(fichero1);
    fwrite(&texto,strlen(texto),fichero1);
    fixero.longitud[idtext]=ftell(fichero1);
    texto=" Esto es otro$ejemplo de $dialogos :)$";
    idtext++;
    Fixero.inicio[idtext]=ftell(fichero1);
    fwrite(&texto,strlen(texto),fichero1);
    fixero.longitud[idtext]=ftell(fichero1)-fixero.inicio[idtext];
    fclose(fichero1);
    save("a.maq",offset fixero, sizeof(fixero) );
    compress_file("datos.maq");
    encode_file("datos.maq","MAQUI");
end
```

```
    Begin
        load("a.maq",&fixero);

decode_file("datos.maq","MAQUI");

uncompress_file("datos.maq");

fichero1=fopen("datos.maq","r");

fseek(fichero1,fixero.inicio[idtext],
seek_set);

fread(&a,fixero.longitud[idtext]-
1,fichero1);
        fclose(fichero1);

    compress_file("datos.maq");

encode_file("datos.maq","MAQU
I");

    put_text(a, 100, 100, 10,
10);
    End

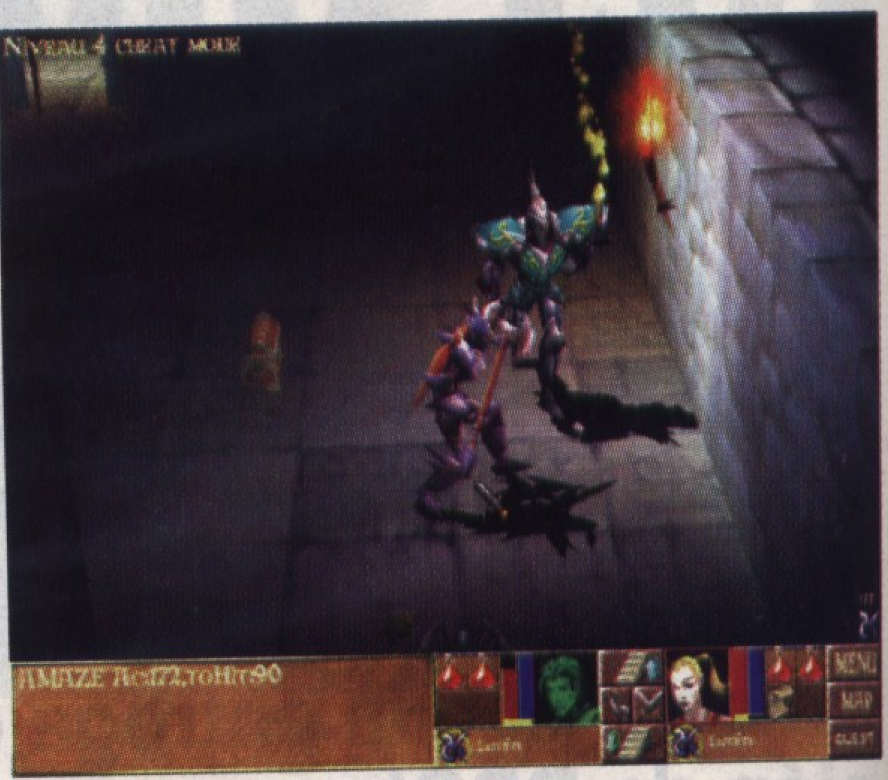
/*
    PARAMETROS:
     - TEXTO -> CADENA DE
TEXTO QUE TIENE QUE SER
VISUALIZADA
     - TX -> POSICION X PARA
EL TEXTO
- TY -> POSICION Y PARA EL
TEXTO
- EL -> ESPACIO ENTRE LINEAS
DE TEXTO
- DELAY -> TIEMPO DE ESPERA
PARA PODER PASAR DE LINEA
*/

    Function put_text(string
texto[5000], int tx, int ty, int el,
int delay);
    Private
        string l1[5000];
        string l2[5000];
        tabla[100];
        i;
        k=1;
        l;
        fin;
        fondo;
        primeras_lineas;
    Begin
        fondo=load_pcx("c:\jue-
gos\dialogo2\b.pcx");
        graph=fondo;
        x=tx+65;
        y=ty+8;

for(size=0;size<50;size+=10);
            frame(300);
        end
        for(i=0;i<5000;i++)
            if(texto[i]=="$")
                tabla[k]=i;
                k++;
            end
        end
        k=1;
    Loop
    if(key(_a) or primeras_lineas==0)
        if(fin==1)
```



La longitudde los párrafos debe ser la adecuada.



Se debe cuidar especialmente la posición de los textos.



Los textos deben estar bien representados en la pantalla y tienen que ser suficientemente atractivos para el jugador.



Los diálogos son la esencia de los juegos de rol.



Hay que hablar hasta con los enemigos.



Aunque luego tengamos que acabar con ellos.

```
l1="";
for(;size>-1;size-=10)
 frame(300);
 end
unload_pcx(fondo);
```

```
 signal(id,s_kill);
else
 l1="";
 l2="";
 l=k-1;
```

```
for(i=0;i<tabla[k];i++)
 strcat(l1,texto[i]);
end
if(strlen(l1)!=0)
strdel(l1,tabla[l]+1,0);
```

Cuadro 2. Código ejemplo 2.

```
Program Diálogos;
Begin
 set_mode(m320x240);
 select_text(0);
 select_text(1);
End

/*
 PARAMETROS:
 - IDTEXT -> NUMERO DEL
TEXTO QUE QUEREMOS ESCRIBIR
 */

Function select_text(int idtext)
Private
 fichero1;
 string a[5000];
 struct fixero
  inicio[10];
  longitud[10];
 end
Begin
 load("a.maq",&fixero);
decode_file("datos.maq","MAQUI");
uncompress_file("datos.maq");
fichero1=fopen("datos.maq","r");
fseek(fichero1,fixero.inicio[idtext],seek_set);
fread(&a,fixero.longitud[idtext]-1,fichero1);
 fclose(fichero1);
 compress_file("datos.maq");
encode_file("datos.maq","MAQUI");
 put_text(a, 100, 100, 10, 10);
End

/*
 PARAMETROS:
 - TEXTO -> CADENA DE
TEXTO QUE TIENE QUE SER
VISUALIZADA
 - TX -> POSICION X PARA EL
TEXTO
 - TY -> POSICION Y PARA EL
TEXTO
 - EL -> ESPACIO ENTRE LINEAS DE
TEXTO
 - DELAY -> TIEMPO DE ESPERA
PARA PODER PASAR DE LINEA
 */

 Function put_text(string texto
[5000], int tx, int ty, int el, int delay);
 Private
  string l1[5000];
  string l2[5000];
  tabla[100];
  i;
  k=1;
  l;
  fin;
  fondo;
  primeras_lineas;
 Begin
fondo=load_pcx("c:\juegos\dialogo2\b.pcx");
  graph=fondo;
  x=tx+65;
  y=ty+8;
 for(size=0;size<50;size+=10);
   frame(300);
  end
  for(i=0;i<5000;i++)
   if(texto[i]=="$")
    tabla[k]=i;
    k++;
   end
  end
  k=1;
 Loop
  if(key(_a) or primeras_lineas==0)
   if(fin==1)
    l1="";
    for(;size>-1;size-=10)
     frame(300);
    end
    unload_pcx(fondo);
    signal(id,s_kill);
   else
    l1="";
    l2="";
    l=k-1;
    for(i=0;i<tabla[k];i++)
     strcat(l1,texto[i]);
    end
    if(strlen(l1)!=0)
   strdel(l1,tabla[l]+1,0);
    else
     l1="";
     for(;size>-1;size-=10)
      frame(300);
     end
     unload_pcx(fondo);
     signal(id,s_kill);
    end
    k++;
    l++;
    for(i=0;i<tabla[k];i++)
     strcat(l2,texto[i]);
    end
    if(strlen(l2)!=0)
   strdel(l2,tabla[l]+1,0);
    else
     fin=1;
    end
    write(0,tx,ty,0,l1);
    write(0,tx,ty+el,0,l2);
    for(i=0;i<delay;i++)
     frame;
    end
    k++;
    primeras_lineas=1;
    frame;
   end
  end
```



Hay que procurar siempre que las conversaciones no sean demasiado farragosas y extensas.

```
        else
            l1="";
            for(;size>-1;size-=10)
                frame(300);
            end
unload_pcx(fondo);
        signal(id,s_kill);
        end
        k++;
        l++;
for(i=0;i<tabla[k];i++)
            strcat(l2,texto[i]);
        end
        if(strlen(l2)!=0)
strdel(l2,tabla[l]+1,0);
        else
            fin=1;
        end
        write(0,tx,ty,0,l1);
        write(0,tx,ty+el,0,l2);
        for(i=0;i<delay;i++)
            frame;
        end
            k++;
            primeras_lineas=1;
            frame;
        end
    end
    frame;
  End
End
```

¿Qué hacen los programas?

El objetivo del primer programa es almacenar los textos en un archivo. Para ello utiliza dos archivos, uno que contiene los textos en sí (que será comprimido y codificado) y otro que indica la posición inicial y longitud de cada uno de los textos.

Los textos introducidos tendrán que tener el carácter "$" para indicar que se trata del final de línea.

El segundo programa es el encargado de leer los datos desde los ficheros que previamente hemos grabado. Una vez se recogen los textos, estos se muestran de dos líneas en dos líneas (el final de la línea lo marca el carácter "$") y con un marco de fondo.

Comentario del código

• Programa 1:

En el programa se usan algunas de las nuevas funciones del DIV2 para el tratamiento de ficheros, concretamente *fopen* para abrir el archivo y *fwrite* para escribir en él.

El primer fichero que se graba es el que contiene los textos (*datos.maq*).

El segundo fichero graba una estructura (*struct fixero*); en esta estructura se graban la longitud y punto de inicio de los textos.

• Programa 2:

Program diálogos

Esta parte del programa se encarga simplemente de poner el modo gráfico y llamar a la función *select_text*.

Function *select_text*(*idtext*)

Esta función recibe como parámetro la variable *idtext*, que nos servirá para sacar el texto del fichero previamente grabado.

Las funciones usadas para obtener el texto son *fseek*, que nos permitirá poner el cursor en la posición que nos marca la estructura (struct *fixero*), y *fread* para leer todos los caracteres del texto (la longitud del texto viene apuntada en la variable *fixero.longitud[idtext]*).

Más tarde se cierra el fichero con *fclose*, se comprime y se codifica.

Para finalizar se llama a la función *put_text* que es la que se encarga propiamente de representar el texto.

Function put_text(string texto[5000], int tx, int ty, int el, int delay)

Esta función recibe 5 parámetros, que son respectivamente: el texto a escribir, la posición x del texto, la posición y del texto, el espacio entre líneas del texto y el tiempo que hace falta esperar para poder pasar a las siguientes líneas de texto.

Para poder dividir el *string* en líneas primero se crea una tabla que marcará las posiciones de los "$" que son los que marcan el final de línea.

Una vez que ya sabemos las posiciones en las que se acaba la línea, creamos las dos primeras líneas y las escribimos. Para avanzar a las siguientes líneas sólo se tiene que pulsar la "A", cosa que hará repetir el proceso de creación de líneas pero empezando por la anterior y no desde el principio como habíamos hecho antes.

Ramón de España (MAQNAZILLER)



En los juegos de rol hay que interactuar con los personajes.

Fasttracker 2.08 (3)

Más allá de la música electrónica

Con Fasttracker podremos grabar todo tipo de composiciones musicales para luego introducirlas en el juego que estemos desarrollando, así lograremos recrear el ambiente adecuado a cada situación.

En los dos números anteriores tratamos todo lo relativo sobre el manejo de *Fasttracker*. Si bien no se profundizó en absolutamente todas las opciones del programa, sí se hizo hincapié en aquellas que resultarán indispensables para lograr los objetivos que se van a plantear en este artículo.

Mejoras de la interfaz

Introducimos este pequeño inciso para decir a todos aquellos usuarios de DIV, que hayáis estado siguiendo con interés los últimos artículos de esta sección, que estamos de enhorabuena, porque la segunda versión de DIV admite más formatos sonoros que antes. Ya no sólo podemos incluir archivos WAV (PCM para ser exactos), sino también diferentes formatos de los secuenciadores más conocidos, tales como el que ahora nos va a ocupar.

> Actualmente, el proceso de elaboración de la música de un juego de ordenador no difiere mucho de la grabación de un disco profesional

Gracias a esto, podrá aumentar la calidad de muestreo de nuestras creaciones, pues ya no tendremos que mirar tanto por economizar memoria y, así mismo, podrán ser de mayor duración. Para ser exactos reducimos aproximadamente a 1/6 la cantidad de memoria necesaria para almacenar la música, dependiendo, claro está, de la cantidad de "samples" de que hagamos uso en nuestros módulos.

Cómo se hace hoy dia

Con lo visto hasta ahora, somos capaces de lograr composiciones bien complejas, sin límite en cuanto a técnica musical se refiere. Incluso los resultados podrán ser de gran calidad sonora si se dispone de una librería de instrumentos (archivos de extensión XI) suficientemente nutrida y cuidada.

Pero la misión de la música y una de sus más estimables cualidades, es su capacidad de transmitir sentimientos. Sería posible reproducir mediante nuestro programa con exactitud matemática *"Las cuatro estaciones"* de Vivaldi (bastaría con poseer las partituras y transcribirlas al formato tabulado de *Fasttracker*), pero seguramente no podríamos llegar a transmitir el "feeling" que es inherente a las flautas que aparecen en la pieza correspondiente a la estación primaveral.

Y es que todo esto ha formado parte de la evolución de la música en los programas de ordenador. Hace unos pocos años, en el tiempo de apogeo de las máquinas de 16 bit Atari y Amiga, y poco después en los equipos PC, sólo se estilaban dos formas para poner música a los videojuegos, mediante MIDI en programas del tipo aventura gráfica (sólo en PC) o mediante módulos realizados con secuenciadores como *Fasttracker*, sobre todo en juegos arcade y de plataformas. El resultado del MIDI deja mucho que desear, puesto que es mucho más rígido que los secuenciadores, y además depende mucho de la tarjeta de sonido que se está utilizando, si bien es cierto que ocupa poca memoria, pero en todo caso es imposible añadir voces e infinidad de sonidos que el ordenador no puede imitar eficazmente. Lo que está claro es que si se desea un resultado realmente impactante no podemos limitarnos por problemas de memoria. Por eso, cuando los ordenadores han tenido potencia suficiente, ha cambiado sustancialmente la forma de hacer las cosas.

Actualmente, los videojuegos son auténticas superproducciones, para las cuales se componen estupendas bandas sonoras, que son grabadas por músicos profesionales con instrumentos reales, en estudios de grabación equipados con la más moderna tecnología. Los resultados obtenidos, que perfectamente podrían pasar a estar en las estanterías de cualquier tienda de discos, son sencillamente sampleados con una alta calidad y utilizados en formato WAV para dichos juegos.

Fases del método de producción semiprofesional con Fasttracker y elementos que toman parte en cada una de ellas

Fase	Ordenador 1	Ordenador 2	Amplificador	Multiefectos	Instrumentos
Preparación del módulo base	Sí	No	No	No	No
Sampleo del módulo base	Sí	Sí	No	No	No
Grabación	Sí	Sí	Sí	Sí	Sí
Producción	Sí	No	No	No	No

Cómo hacerlo

Seguro que en estos momentos habrá más de un lector cruzado de brazos, frustrado por la idea de que no va a tener jamás acceso a un estudio profesional de grabación, ni a los conocimientos necesarios para manipular toda esa parafernalia. Pues bien, que nadie se preocupe. A continuación vamos a detallar un método por el cual, si disponemos de los músicos que necesitamos, y algo de equipo, podremos llegar a obtener resultados realmente sorprendentes.

Con poco más equipo que dos ordenadores podremos montar un estudio de grabación en nuestra propia casa

Supongamos que queremos ponerle música a un sanguinario arcade ambientado en la edad media. Para ello optamos por una banda sonora de estilo "heavy metal". Los lectores a los que les guste este tipo de música, y ya hayan intentado realizar alguna composición con *Fasttracker*, se habrán encontrado con las dificultades que surgen a la hora de emular solos de guitarra, que para estos estilos musicales son primordiales. Por eso creo que éste resultará ser un ejemplo muy útil.

Qué necesitamos

Lo principal: necesitamos un guitarrista y un vocalista. Si queremos meter bajos con técnicas modernas como "slapping" o "tapping", también necesitaremos un bajista. Podremos prescindir de teclista o batería, ya que estos instrumentos podrán ser emulados muy eficazmente secuenciándolos, y ahorrando así horas de trabajo e incomodidades (es muy complicado grabar una batería). Por supuesto, si se quieren introducir flautas o saxos habría que encontrar músicos que interpretaran esas partes.

En cuanto a la parte técnica, el equipo básico comprendería los siguientes elementos:

Dos ordenadores con tarjeta de sonido: uno de ellos deberá ser al menos un Pentium 120 con 64 Mb de RAM y con una tarjeta de sonido de calidad (*GRAVIS ULTRASOUND o SOUNDBLASTER AWE*). En cuanto al otro bastará con que sea un 486 ó superior.

Amplificadores para los instrumentos que se utilicen, para que la salida que se obtenga esté preamplificada y tratada con circuitos diseñados para ese fin.

Generadores de efectos para los instrumentos, sobre todo para las distorsiones de las guitarras, que han de sonar lo más profesional y contundente posible. También sería interesante disponer de un módulo de "reverb" para la voz, aunque ésta la podemos conseguir mediante los efectos de *Fasttracker*.

Cables "jack-jack" tamaño grande para interconectar los diferentes aparatos e instrumentos y un adaptador para conectar la salida final al ordenador por la entrada "line-in2 de la tarjeta de sonido.

Un cable "jack-jack" pequeño para conectar ambos ordenadores entre sí.

Fasttracker instalado en ambos ordenadores.

Empezamos

Lo primero que vamos a aclarar es, ¿para qué se necesitan dos ordenadores? Es sencillo, uno de ellos (el más potente) será el que se encargará de samplear los fragmentos que interpreten los músicos, y será en el que se vaya construyendo el módulo final. A éste le llamaremos ordenador 1. El otro ordenador (ordenador 2) se encargará de servir como metrónomo a los músicos, es decir, mientras los músicos graban, el ordenador 2 estará reproduciendo el acompañamiento de batería y bajo, para dar el tiempo y el tono de la canción.

Visto así, parece sencillo. Bastaría con tener preparado un módulo en el ordenador 2 con la batería y el bajo, el cual pondríamos a reproducirse mientras sampleamos con el ordenador 1. Pero no, desafortunadamente un mismo módulo reproducido en dos ordenadores distintos no va exactamente a la misma velocidad, lo cual representa un serio problema. La solución es la siguiente:

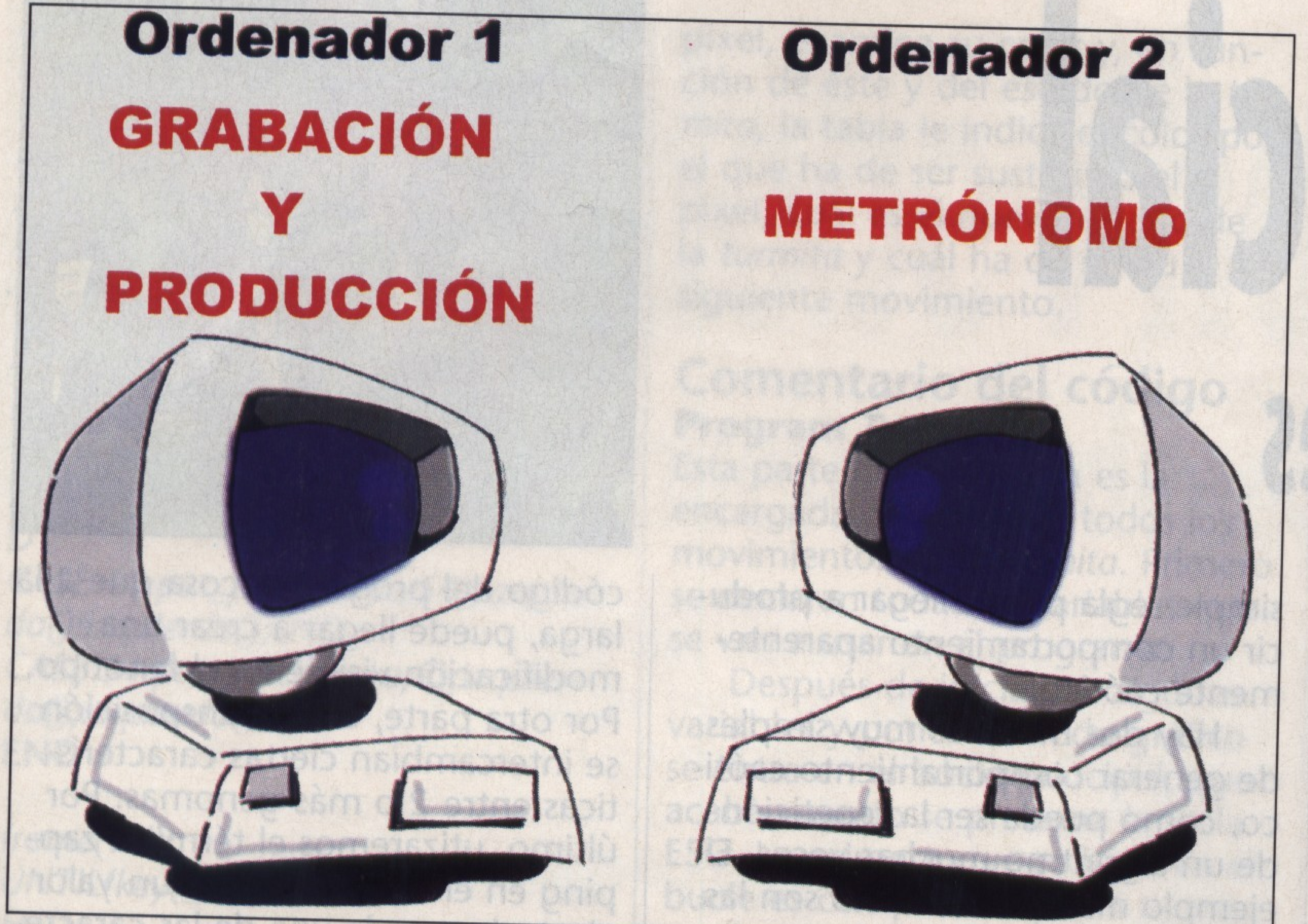
Prepararemos un módulo con la batería y el bajo de la canción en el ordenador 1.

Conectamos la salida de "line-out" del ordenador 1 al "line-in" del ordenador 2.

Sampleamos lo que sale del ordenador 1 con el ordenador 2, eso sí, con una calidad baja de muestreo, ya que en el ordenador 2 dispondremos de poca memoria.



Elementos que conectamos al ordenador 1 en la fase de grabación.



El cometido de cada uno de los dos ordenadores es distinto y no es intercambiable.

En teoría ya debería estar todo solucionado. Pero si reproducimos el "sample" en el ordenador 2 y vemos que tampoco está perfectamente sincronizado con el módulo que creamos en el ordenador 1, no debemos asustarnos. Esto se debe al hecho de que al samplear con *Fasttracker* se produce un leve desajuste en la frecuencia. Sólo hay que hacerle una pequeña modificación al "sample" que ahora mismo tenemos en el ordenador 2. Entramos en el editor de instrumentos ("Instr. Ed.") y hacemos disminuir en, digamos, 8 unidades, la variable "Tune", situada en la parte derecha de la pantalla.

Una vez que hayamos sampleado todos los fragmentos que nos van a servir de metrónomo para realizar la grabación, quitaremos el cable que conectaba los dos ordenadores para realizar el siguiente paso.

Grabando

Es la hora de grabar. Empecemos por ejemplo con la guitarra. Conectamos la guitarra al procesador de efectos, el procesador de efectos al amplificador, y por último, la salida del amplificador al "line-in" del ordenador 1. Colocamos el volumen de la fuente (el amplificador) a un volumen óptimo para que se oiga lo suficiente sin producir distorsiones. Ponemos a reproducir en el ordenador 2 el "sample" que corresponda al fragmento que queramos grabar y en el ordenador 1 entramos en el "Smp.Ed." de *Fasttracker* y nos ponemos a samplear. Lo ideal será que grabemos con la máxima frecuencia de muestreo y en 16 bit para que el sonido obtenido tenga calidad CD, por eso es conveniente que el ordenador 1 tenga la mayor cantidad de memoria RAM posible, ya que los fragmentos de cada instrumento ocuparán un espacio muy considerable. Repetimos tantas veces como sea preciso hasta que quedemos satisfechos con la toma, que luego será tratada como si de un instrumento más se tratase (extensión XI), sólo que en este caso lo reproduciremos una sola vez al principio de cada fragmento para que emerja todo un fraseo completo.

Es importante conseguir un buen sonido de guitarra, por ello no debemos escatimar tiempo en las pruebas previas de ecualización. Hay que tener en cuenta que las tarjetas de sonido que no son totalmente profesionales, dan salidas pobres en graves y debemos compensarlo en la grabación, porque en el proceso de producción ya quedará poco que hacer al respecto.

Para la grabación de la voz las conexiones son más sencillas. Bastaría con conectar un micrófono de baja impedancia (para evitar acoples) a la entrada "mic-in" de la tarjeta de sonido del ordenador 1. También podría conectarse el micrófono al ordenador a través de un procesador de efectos para dotar de "reverb" a la voz, con lo cual se conseguiría que los resultados finales fueran más naturales. De tomar esta última opción deberíamos conectar el procesador de efectos a la entrada "line-in" y no a la "mic-in". Si no utilizamos un multiefectos podremos ponerle "reverb" a la voz en el proceso de producción.

Un truco: grabar dos veces la misma línea vocal para luego reproducirlas ambas a la vez, puede ser muy socorrido para limar imperfecciones.

Proceso de producción

Ya tenemos todo el material necesario grabado. Debemos ubicar entonces todos los "samples" que hemos obtenido en el módulo que sirvió de metrónomo para hacer las grabaciones. Ahora es un buen momento para añadir teclados, si así lo deseamos, secuenciándolos.

El proceso de equilibración de volúmenes y producción es esencial, porque es lo que realmente dará buen o mal sonido al resultado final. En caso de que queramos subir el volumen de algo que ya no de más de sí con el control de volumen del editor de instrumentos, debemos hacerlo reproduciendo el mismo "sample" en otro canal al mismo tiempo, y nunc subiendo el volumen desde el editor de "samples", ya que esto produciría distorsiones. Podremos conseguir "delay" o "reverb" reproduciendo el mismo "sample" en varios canales haciéndolos empezar paulatinamente más tarde y con un volumen inferior. Así mismo, se podría poner un efecto de "chorus" a un "sample" reproduciéndolo a la vez en otro canal con "vibrato".

El proceso de producción será el que mayormente determinará la calidad del resultado final

Una vez terminado el proceso de producción tan sólo queda elegir el formato final de almacenamiento, dependiendo de los requerimientos de memoria.

Sergio Cánovas

CONCLUSIONES

En estos tres números hemos llegado a manejar la inmensa mayoría de las opciones que *Fasttracker* nos ofrece, hasta el punto de llegar a utilizarlo, no sólo como secuenciador, sino como mezcladora y procesadora de efectos para la grabación de sonido semiprofesional.El proceso explicado en este número ha sido utilizado para la grabación del trabajo "Soberbia, opulencia y lascivia" del grupo *La Orden del Kaos*, habiendo obtenido óptimos resultados.

Vida artificial

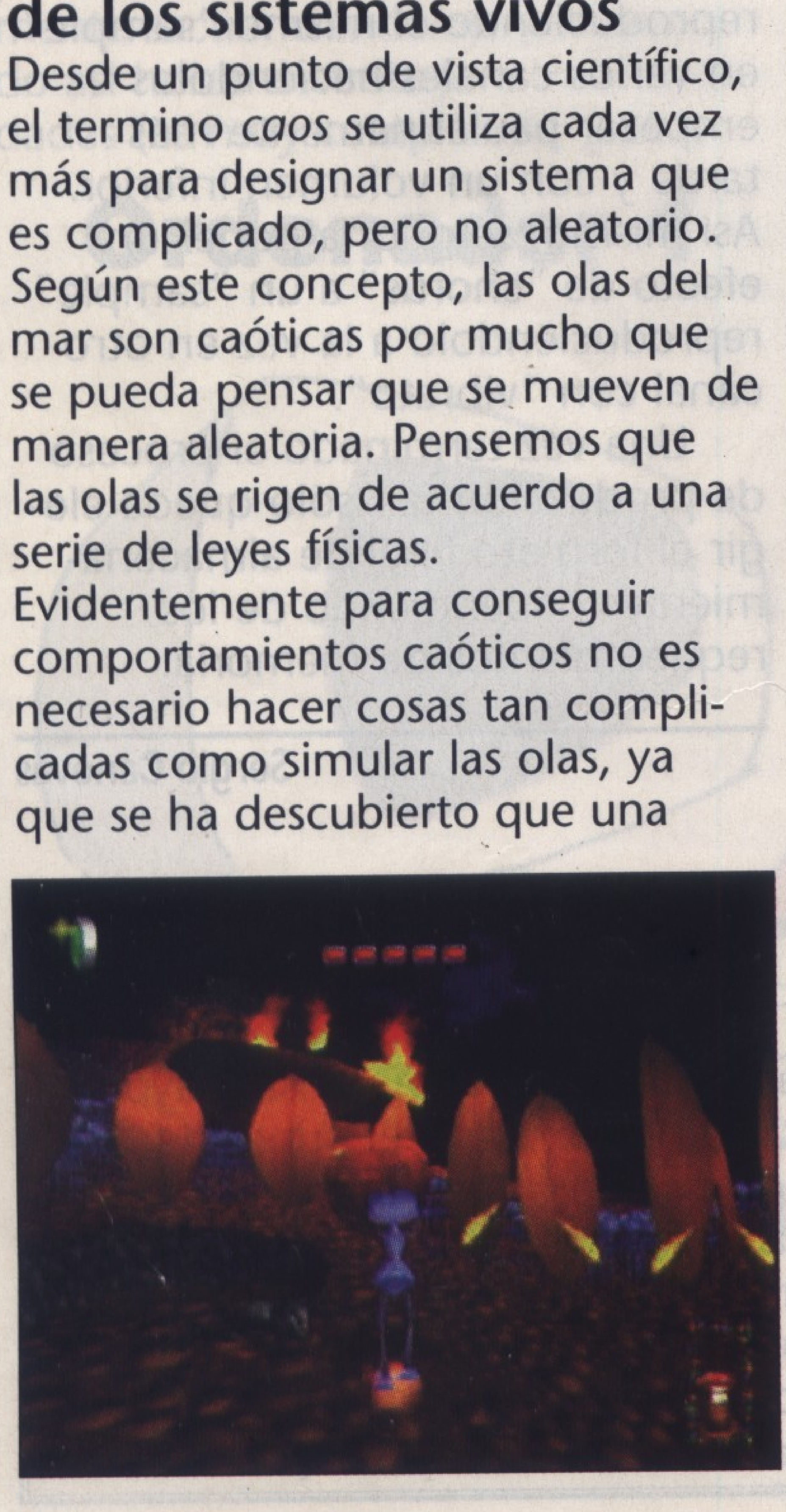
Jugando a ser dioses

Para facilitar la comprensión del artículo lo trataremos desde dos puntos de vista; el teórico y el práctico. En el apartado teórico explicaremos los conceptos más interesantes de la Vida Artificial y en el práctico veremos dos ejemplos simples programados en DIV.

De manera genérica podemos definir la Vida Artificial (VA a partir de ahora) como la ciencia que estudia la creación de sistemas hechos por el hombre que se comportan como si estuviesen vivos. Debemos distinguir los conceptos VA y IA: mientras que el primero supone la creación de sistemas complejos mediante el uso de unas reglas simples, el segundo indica la creación de sistemas simples mediante gran número de reglas o reglas complejas.

Características de los sistemas vivos

Desde un punto de vista científico, el termino *caos* se utiliza cada vez más para designar un sistema que es complicado, pero no aleatorio. Según este concepto, las olas del mar son caóticas por mucho que se pueda pensar que se mueven de manera aleatoria. Pensemos que las olas se rigen de acuerdo a una serie de leyes físicas. Evidentemente para conseguir comportamientos caóticos no es necesario hacer cosas tan complicadas como simular las olas, ya que se ha descubierto que una simple regla puede llegar a producir un comportamiento aparentemente caótico.



Hay dos maneras muy simples de generar comportamiento caótico, como puede ser la repetición de un algoritmo muchas veces. El ejemplo más claro de esto son las leyes de Mendel. También puede serlo la creación de un entorno donde las copias de un mismo algoritmo pueden interactuar.

Sexo

Dentro de este apartado distinguiremos 4 aspectos: Genes, Genoma (el conjunto de cromosomas que contienen la información de un organismo), Fenotipo (los caracteres hereditarios externos que dependen del genoma) y reproducción. Todas las criaturas nacen con un genoma, y éste se encarga de construir el fenotipo. En una simulación de VA en ordenador podemos decir que el genoma es el programa y el fenotipo la representación gráfica. Dado que el fenotipo aparece a partir de un pequeño genoma resulta muy sencillo hacer copias a partir del genoma, sin necesidad de duplicar el complicado fenotipo. En las representaciones por ordenador esto se realiza de una manera muy mecánica, es decir, se duplica el código (genoma) haciendo aparecer 2 fenotipos. Mediante el apareamiento combinaremos el genoma de los padres mediante un proceso bastante complejo.

Cambios genéticos

Podemos encontrar 3 tipos: Mutación, Transposición y Zapping.

En la vida real las mutaciones se producen a consecuencia de ciertos productos químicos, radioactividad y otros agentes. Algunas mutaciones pueden ser letales, pero la mayoría de ellas no muestran ninguna diferencia del todo visible. Puede ser una mutación o la acumulación de ellas las que pueden modificar el aspecto o el comportamiento del fenotipo. En las representaciones con ordenador la mutación es algún cambio en el



código del programa, cosa que a la larga, puede llegar a crear una modificación visible en el fenotipo. Por otra parte, en la transposición se intercambian ciertas características entre 2 o más genomas. Por último, utizaremos el término zapping en el sentido de dar un valor aleatorio a cada una de las características del genoma. En el mundo real el zapping no es un método viable de modificación genética, ya que tras su aplicación la criatura producida muere instantáneamente. Este método es interesante para las representaciones por ordenador, ya que aquí no hay riesgo de que la criatura muera.

Muerte

La muerte está presente en todos los sistemas con seres vivos, incluso en los entornos creados por un ordenador. La muerte desde un punto de vista personal es terrible, pero para la evolución es necesaria, y sin evolución nosotros no hubiéramos existido.

Para que haya evolución son necesarias 3 factores: reproducción, variación del genoma y selección natural. Es evidente que para que exista la selección natural tiene que haber muerte.

A continuación veremos dos ejemplos: el primero es una *turmita*, y el segundo una hormiga.

Ejemplo 1: Turmita

```
Program TURMITA;

GLOBAL
orientacion=1;
xa;
ya;
vel=10;
a=1;
BEGIN
set_mode(m320x240);
xa=160;
ya=120;
write(0,0,0,0,"Pulsa A para aumentar la velocidad");
write(0,0,10,0,"Pulsa D para disminuir la velocidad");

REPEAT
IF(xa<0)xa=320;END
```

```
IF(xa>320)xa=0;END
IF(ya<0)ya=240;END
IF(ya>240)ya=0;END
IF(key(_a))vel—
;IF(vel<1)vel=1;END;END
IF(key(_d))vel++;END

SWITCH(get_pixel(xa,ya))
CASE 0:put_pixel(xa,ya,22);dere-
cha();END
                    CASE
22:put_pixel(xa,ya,41);derecha();EN
D
CASE 41:put_pixel(xa,ya,54);izquier-
da();izquierda();END
CASE 54:put_pixel(xa,ya,0);izquier-
da();izquierda();END
END

frame(vel);
UNTIL(key(_esc))
END

////////////////////////////////////

function derecha();

BEGIN
orientacion++;
IF(orientacion>4)orientacion=1;END

SWITCH(orientacion)
CASE 1:xa-=a;END
CASE 2:ya+=a;END
CASE 3:xa+=a;END
CASE 4:ya-=a;END
END
END

////////////////////////////////////

function izquierda();
BEGIN
orientacion—;
IF(orientacion<1)orientacion=4;END

SWITCH(orientacion)
CASE 1:xa+=a;END
CASE 2:ya-=a;END
CASE 3:xa-=a;END
CASE 4:ya+=a;END
END
END
```

¿Qué es y qué hace una Turmita?

Una *turmita* es una máquina de Turing n-dimensional. El nombre de *turmita* proviene de la yuxtaposición de las palabras Turk (fue quien hizo la primera *turmita*) y termita.

El universo de las *turmitas* se compone de una superficie pintada con diferentes colores.

Las *turmitas* se mueven sobre la superficie de acuerdo a las instrucciones de una tabla que se comporta igual que las maquinas de Turing convencionales. Cuando una *Turmita* se sitúa encima de un pixel, examina su color y, en función de éste y del estado de la *turmita*, la tabla le indica el color por el que ha de ser sustituido el pixel, cuál es el nuevo estado de la *turmita* y cuál ha de ser su siguiente movimiento.

Comentario del código

Program *Turmita*

Esta parte del programa es la encargada de controlar todos los movimientos de la *turmita*. Primero se declaran todas las variables que se van a usar en el programa.

Después de inicializar algunas variables y poner el modo gráfico se entra en el bucle principal, que acaba cuando se presiona la tecla ESC. Lo primero que se hace en el bucle es comprobar que la *turmita* no se salga de los limites y luego se comprueba de qué color es el pixel en el que esta situado. En función del color obtenido se llama a la función derecha o izquierda y se pone el pixel de otro color. Observad que a la función izquierda se le llama dos veces. Esto es así porque esta combinación (derecha/derecha/izquierda, izquierda/izquierda7izquierda) da lugar a un resultado bastante interesante. Podéis probar otras combinaciones para obtener diferentes resultados.

Function derecha

Esta función aumenta en 1 la orientación y comprueba que no pase de 4. Después comprueba el valor de la orientación y modifica la posición x o y de la *turmita*, según el valor de la variable posición.

Function izquierda

Hace lo mismo que la función derecha pero a la inversa.

Ejemplo 2: Hormiga

```
Program HORMIGA;

Global

  noa=5000;
  coorx;
  coory;
  coorx0;
  coory0;
  inx;
  iny;
  azul=54;
  rojo=22;
  negro=0;
  pixel;
  pixel0;
  carga=0;
  pos;
  vel=100;

Begin
```



```
  set_mode(m320x240);
  empieza();
  write(0,0,0,0,"Pulsa A para
aumentar la velocidad de la hormi-
ga");
  write(0,0,10,0,"Pulsa D para dis-
minuir la velocidad de la hormiga");

Loop

    actualiza();
    inx=rand(-3,3);
    iny=rand(-3,3);

pixel=get_pixel(coorx+inx,coory+iny);

    if(coory<0)
       coory=0;
       iny=-iny;
    end

    if(coory>240)
       coory=240;
       iny=-iny;
    end

    if(coorx<0)
       coorx=0;
       inx=-inx;
    end

    if(coorx>320)
       coorx=320;
       inx=-inx;
    end

    if(KEY(_a))
       vel—;
       if(vel<1)
          vel=1;
       end;
    end

    if(key(_d))
       vel++;
    end

    coorx+=inx;
    coory+=iny;

    if(pixel==negro and carga==azul)
       descargar();
    end

    if(pixel==azul and carga==negro)
       cargar();
    end

    if(pos==0)
```

```
        put_pixel(coorx0,coory0,negro);
    else
        put_pixel(coorx0,coory0,azul);
    end

    if(get_pixel(coorx,coory)==azul)
        pos=1;
    else
        pos=0;
    end

    put_pixel(coorx,coory,rojo);

    frame(vel);

  end

  END

//////////////////////////////////

Function empieza();

Private
i;

Begin

  for(i=0;i<noa;i++)

  put_pixel(rand(0,320),rand(0,240),a
  zul);
    end

    coorx=rand(0,320);
    coory=rand(0,240);

end

//////////////////////////////////

Function actualiza();

Begin

  coorx0=coorx;
  coory0=coory;
  pixel0=pixel;

END

//////////////////////////////////

Function descargar();

Private

  i,j,suma;

Begin

  for(i=coorx-1;i<=coorx+1;i++)
    for(j=coory-1;j<=coory+1;j++)
      if(get_pixel(i,j)==azul)
        suma++;
      end
    end
  end

  if(suma>2)
      carga=negro;
      pixel=azul;
      put_pixel(coorx,coory,azul);
  end

End

//////////////////////////////////

Function cargar();

Private

  i,j,suma;

Begin

  for(i=coorx-1;i<=coorx+1;i++)
    for(j=coory-1;j<=coory+1;j++)
      if(get_pixel(i,j)==negro)
        suma++;
      end
    end
  end

  if(suma>6)
    carga=azul;
    pixel=negro;
    put_pixel(coorx,coory,negro);
  end

End
```

¿Qué hace la Hormiga?

Este programa llena la pantalla de puntos azules (que nosotros imaginaremos que es la comida de la hormiga). La hormiga, que es un pixel rojo, irá cogiendo los pixels azules (comida) que tengan más de 6 espacios libres a su alrededor, y los depositará en un espacio vacío que a su alrededor tenga al menos 3 espacios ocupados por comida.

Como podéis observar tan sólo hay dos reglas muy simples que permitirán ordenar la comida. Recordad que una de las principales características de la Vida Artificial es conseguir comportamientos complejos mediante reglas simples. En nuestro caso las reglas son: coger los puntos azules que tengan más de 6 espacios vacíos a su alrededor y dejar la comida en un lugar vacío que tenga más de 2 puntos azules a su alrededor. El programa también permite aumentar o disminuir la velocidad mediante el uso del teclado y comprueba que la hormiga no se salga de los límites.

Explicación del código

Program Hormigas

Esta función se encarga de todo el control del programa.

Primero se declaran todas las variables globales para el funcionamiento del programa, se inicializa el



modo gráfico, se ponen los textos en pantalla y se llama a la función *empieza* que comentaremos más adelante. A partir de aquí empieza el bucle principal; éste controla las entradas del teclado para poder aumentar o disminuir la velocidad de la hormiga. Este proceso llama a la función *actualizar* y también se encarga de impedir salir a la hormiga de sus límites. Por ultimo se actualiza la posición de la hormiga, se borra la anterior posición y se comprueba si se cumplen las condiciones para cargar o descargar (para poder descargar tiene que estar cargada, y para poder cargar tiene que estar descargada).

Función empieza()

Esta sencilla función pone los puntos azules mediante un bucle *for* y se encarga de calcular la coordenada (x, y) de inicio de la hormiga.

Función actualiza()

Actualiza el contenido de las variables *coorx0, cooryo* y *pixel0*, que indican respectivamente la posición x, la posición y, y el color del pixel en que la hormiga estaba anteriormente situada.

Función descargar()

Esta función es llamada si la función principal encuentra una posición adecuada para descargar. Lo primero que hace es comprobar el color de los pixels que rodean a la hormiga, y si estos son azules (comida) añade 1 a la variable suma. Si la variable suma es más grande que 2 (cosa que indica que es un lugar apropiado para descargar) se descarga la comida.

Función cargar()

Cuando se llama a esta función se comprueba el color de los pixels colindantes a la hormiga, y si estos son negros se suma 1 a la variable suma. Si la variable suma es más grande que 6 se coge la comida y se borra el pixel azul.

Ramón de España (MaQNaZiLLeR)
http://pagina.de/maqnaziller
maqnaziller@pagina.de

Curso Adobe Photoshop (II)

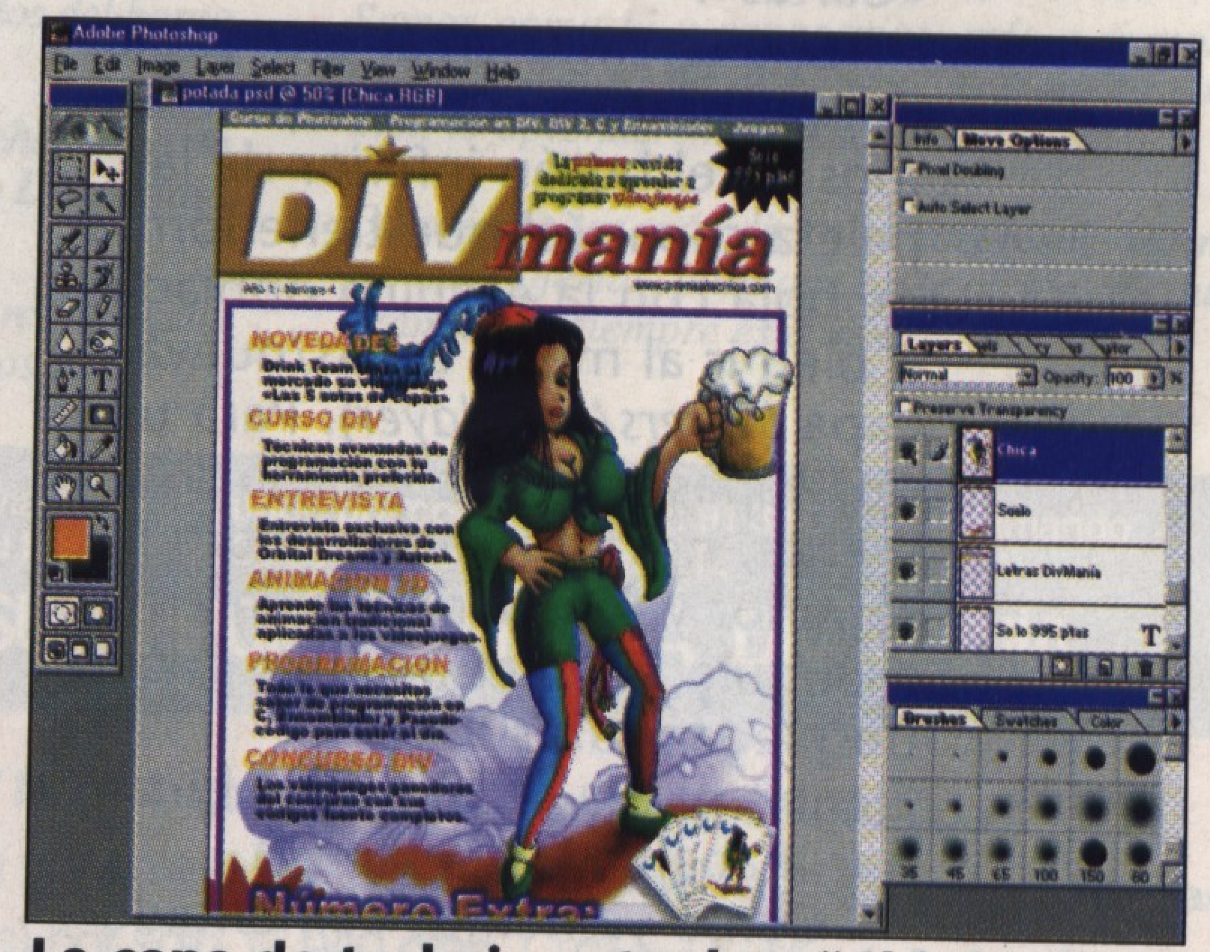
Diseñando en nuestro ordenador

En esta segunda entrega del curso, daremos una introducción a la forma de trabajar con capas (layers) en Photoshop. Las capas son uno de los elementos más potentes con los que cuenta nuestro programa de diseño favorito.

Cuando trabajamos con otros programas de dibujo, que no tienen capas, la opción de dibujar en capas, muchas veces nos encontramos con el problema de querer modificar sólo una parte de la imagen sin alterar el resto. Esta tarea es muy laboriosa cuando tenemos que «recortar» pixel a pixel la parte que queremos modificar y volver a dibujar encima el diseño original. Menos mal que los programadores de Adobe nos brindan la posibilidad de trabajar con capas. Podemos imaginarnos la forma de trabajar en Photoshop con capas si pensamos en un conjunto de cristales transparentes superpuestos. Podemos dibujar en uno o en otro sin alterar los dibujos que haya en el resto. De esta forma, el diseño final resultará de juntar todos los dibujos que haya en los cristales. Esta técnica es la utilizada en los estudios de animación para hacer las películas de dibujos animados. El fondo va por una parte y los personajes se dibujan en láminas de plástico (realmente son hojas de acetato). Cada cuadro (*frame*) que compone la animación resulta de superponer todas las láminas de plástico sobre el fondo (*background*) que es opaco.

¿Qué ventajas tiene el trabajar con capas?

La principal es el ahorro de tiempo en cada diseño y la posibilidad de reutilizar trabajo. En el ejemplo de este artículo, una hipotética portada de DIV Manía, el borde y la cabecera se podrían utilizar para distintos números. Sólo habría que cambiar el dibujo principal de portada y el texto de las secciones.



La capa de trabajo actual es "Chica" y está visible.

Además, en el ajuste de texto y la colocación de los distintos elementos que componen la portada, cuando trabajamos con capas se invierte muy poco tiempo. Los efectos que se pueden añadir al texto (como sombreados, apariencia 3D...) son muy sencillos de realizar. Podrás dibujar, modificar, pegar, mover y utilizar máscaras sobre capas. El nivel de opacidad y el modo de fusión de una capa con otra son también ajustables. En definitiva, todo son ventajas, excepto que una imagen formada por varias capas ocupa más memoria que una imagen formada por una única capa (el *fondo*).



Así es como quedaría la imagen si ocultamos la capa Chica".

Empezando a trabajar

Podéis seguir los pasos de este artículo cargando la imagen del ejemplo que encontraréis en el CD que acompaña a la revista. Está en formato PSD de Adobe Photoshop 5. Una vez cargada la imagen, iréis a la paleta de Capas (Layers) pinchando en su pestaña. Aparecerá una lista de pequeñas imágenes con su nombre correspondiente. A la izquierda de cada imagen, dos cuadros. El cuadro de la izquierda normalmente tendrá un ojo. Esto indica que la capa actual está visible. Probar a pinchar sobre el cuadro del ojo de la capa nombrada como

«Chica». El cuadro de al lado, cuando tenga un pincel pintado, indicará que esa capa es sobre la que estamos trabajando actualmente. Para cambiar de capa activa, bastará con que pinchemos sobre su imagen. Muchas veces resultará útil el cambiar el orden de las capas. Bastará con arrastrar y soltar la capa en la posición de la lista que queramos. Probar el efecto que tiene poner la capa «Suelo» delante de la capa «Cartas».

Si queremos crear una capa nueva, bastará con que pinchemos en el botón inferior de la paleta capas (el que tiene forma de hoja con la esquina doblada) o bien ir al menú capas/nueva capa (*layers/new layer*). Para unir dos capas distintas en una solamente, nos situaremos sobre la superior y pincharemos en el triángulo del menú capas que desplegará un submenú. Elegiremos la opción Combinar hacia abajo (*Merge down*).

Como ya hemos comentado antes, las imágenes con varias capas sólo se pueden salvar en formato PSD, que es original de Photoshop. Cuando tengamos la imagen terminada, si queremos exportarla a otro formato distinto (como BMP, JPG...), tendremos que juntar todas las capas en una sola. Para hacer esto, iremos al submenú comentado antes y seleccionaremos Combinar visibles



Uno de los efectos que hemos aplicado al texto es el sombreado con capas. El resultado es muy bueno.

(*Merge visible*). También podremos crear una capa nueva, duplicar y eliminar una existente desde el mismo submenú (*New layer, Duplicate layer y Delete layer*).

Podemos seleccionar algún elemento de una capa y, al cambiar la capa activa de trabajo, la selección no se borra. Esto abre interesantes posibilidades de trabajo, como el efecto de sombra que hemos utilizado en el texto. Por ejemplo, el texto de la cabecera tiene una sombra degradada de color amarillo. Seleccionaremos la herramienta de texto y al escribirlo no destruiremos la selección. Iremos al menú Selección/Modificar/Expandir y pondremos 5 píxeles. Crearemos una nueva capa, inferior a la del texto, y rellenaremos la selección con un color amarillo claro. Sobre esta capa, aplicaremos tres veces el filtro de Desenfocar Más (*Blur More*). El resultado es el obtenido. Si queréis, podéis usar la herramienta de desenfoque y aplicarlo manualmente.

Una forma de proceder similar es la utilizada para hacer las sombras oscuras desplazadas del texto de la cabecera o de las cartas. Seleccionaremos el objeto, crearemos una nueva capa (y la selección se mantiene). Nos situamos en la nueva capa y movemos la selección unos píxeles más abajo y a la izquierda. Rellenamos la selección con un gris claro y aplicamos el filtro de desenfoque.

La imagen de la chica se realizó con el método de trabajo que explicamos el mes pasado. El resto de la imagen ha sido creada con capas. Podéis modificar las capas de la imagen de ejemplo para aprender más sobre su uso.

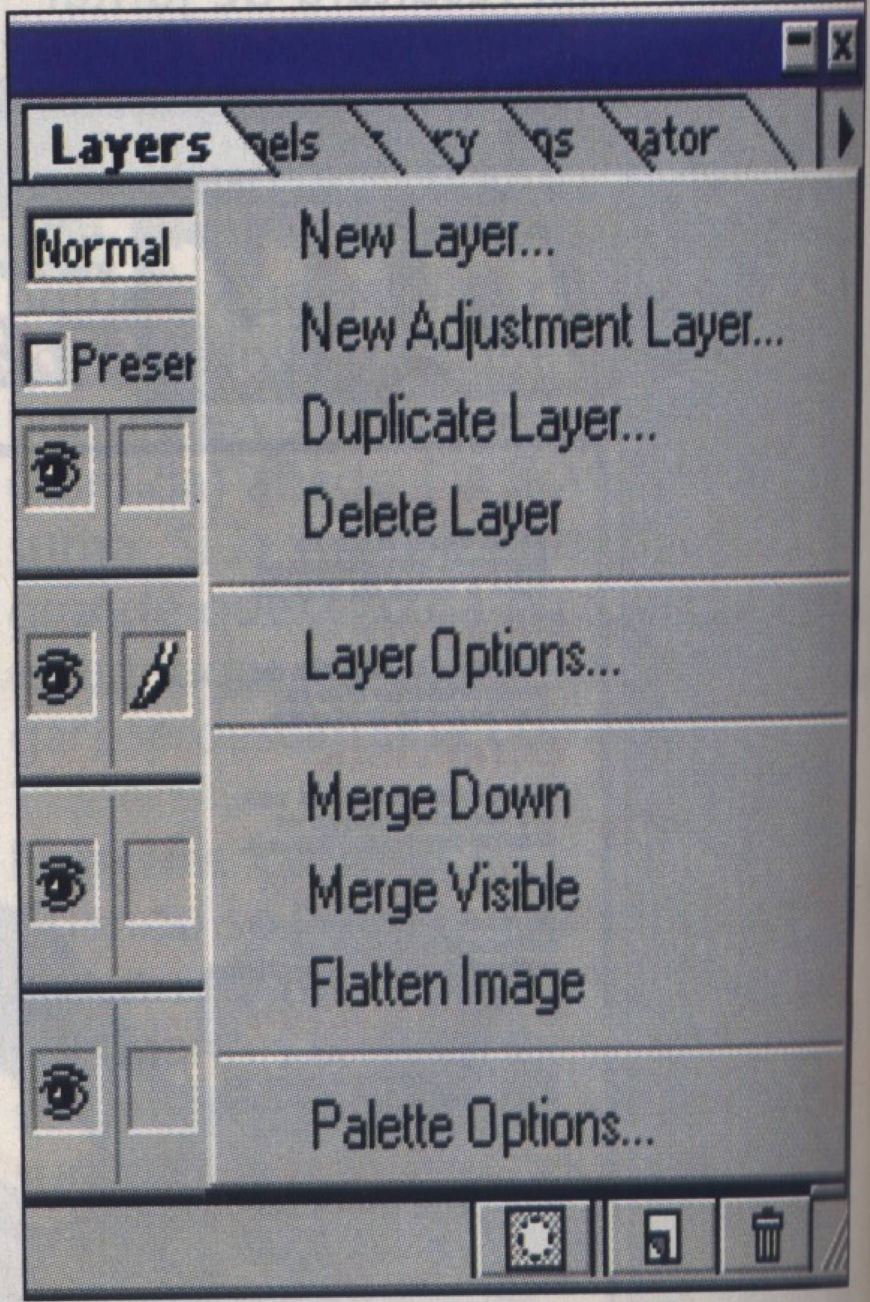
¿Ya hemos acabado con las capas?

Ni mucho menos. El próximo mes abarcaremos lo que nos queda por ver del trabajo con capas, utilización de máscaras de selección y quizás introduzcamos brevemente el manejo de canales. Antes de despedirnos queríamos agradecer a «Las 5 sotas de copas» el permitirnos usar su distintivo para ilustrar este artículo. Nada más por este mes, recibir un saludo desde la redacción.

Carlos Glez. Morcillo



La portada de la revista acabada.



El submenú de la paleta de capas. Para acceder a él, pincharemos sobre el triángulo de la esquina superior derecha.

El correo del lector

Se buscan programadores de videojuegos

Parece que el sector del desarrollo de software lúdico en España está experimentando un fuerte tirón impulsado por el éxito de algunas compañías españolas, que son la punta de lanza de la industria española en cuanto a programas de entretenimiento se refiere. Esperemos que el éxito sea contagioso.

Se están empezando a formar grupos de desarrolladores por todo el territorio nacional. La ayuda que proporcionan los programas tipo *DIV* hace que cada vez sea más sencillo programar títulos propios. Os vamos a mostrar unos cuantos de estos anuncios que nos han llegado a la redacción o que hemos sacado de alguna página de Internet.

Oportunidad de trabajo en una empresa líder

Empresa líder en la edición y distribución de videojuegos busca para sus estudios de producción:

ARTISTAS INFOGRAFISTAS. (Ref: GrafLV5/09).
Se buscan para la creación gráfica de videojuegos.
Requisitos:
- Titulados en Bellas Artes, escuelas de arte, ilustración o cómic.
- Gran creatividad artística.
- Preferiblemente con experiencia en los programas *3D Studio Max* y *Photoshop*.

GAME DESIGNER. (Ref: GdesLV5/09).
Para el diseño de la estructura global de un vídeojuego.
Requisitos:
- Titulados en Informática.
- Facilidad para trabajar en equipo.
- Capacidad para generar nuevos conceptos e ideas en el videojuego.

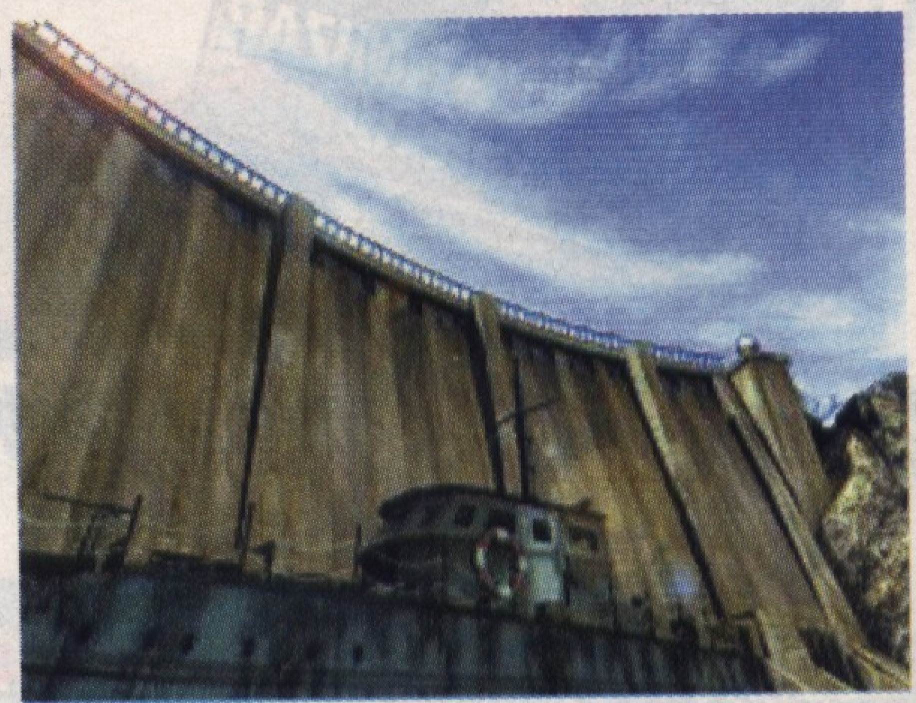
DATA MANAGER. (Ref: DmgerLV5/09).
Para el control y gestión de los datos informáticos de los diferentes proyectos.
Requisitos:
- Titulados en Informática.
- Experiencia en VSS u otros sistemas de organización de bases de datos.
- Dotes de comunicación.
- Persona organizada.

Requisitos comunes a los 3 puestos de trabajo:
- Interesados por los videojuegos.
- Imprescindibles conocimientos de inglés y/o francés.
- Edad: entre 22 y 30 años.

Si te interesa trabajar en una empresa joven, en plena expansión e innovadora, envía tu CV indicando la referencia del puesto de trabajo de tu interés, una carta de motivación y para el puesto de "artista infografista" muestras de trabajo. Todo ello por correo electrónico a empleo@ubisoft.es, por fax al número 93 590 62 53 ó por correo normal a UBI STUDIOS, Crta. De Rubi nº72-74, Edificio Horizon - 08190 Sant Cugat del Vallès (Barcelona).

Nuevos proyectos

Soy un chico de 15 años apasionado por el mundo de la programación, y me gustaría que los programadores aficionados y los grafistas, colaboraran conmigo para llevar a cabo una versión del juego *Racer (Episodio 1)*



para *DIV 1* ó *2*. También se llevarán a cabo otros proyectos tipo *Commandos*.
Sólo pongo la condición de ser menor de 20 años. Gracias.
Dani "Maya"
danis@apabcn.ictnet.es

Se busca

Estoy interesado de adquirir un ordenador PC o Atari STE. Enviar ofertas y dirección para verlo al apartado de correos 274. Talavera de la Reina. Código Postal 45600, (Toledo).

Se abre la tienda

Se venden discos (1.500) de 5"1/4, doble cara, DD. Discos (80) 3" PCW con programas. Discos 3"1/2 con programas. CD-Rom: programas, cursos (informática, inglés, cocina, etc.), enciclopedias, juegos, traductor de inglés. También vendo un modem-fax, una impresora, ratón, máquina de escribir, bici montaña, coche Citröen 2CV. Puedo cambiarlo los artículos por algo que me pueda interesar: mesa mezclas Spirit Folio SX, guitarra Strato, Squier, ARIA Pro II FL 20 o similares, saxo tenor, ordenador Casio FZ1, etapa de potencia, etc. Escribir al apartado de correos 274. Talavera de la Reina (Toledo). Código postal 45600.



Esperamos vuestras cartas

Si quieres enviarnos cualquier comentario o sugerencia, o quieres poner cualquier anuncio, en esta sección de la revista puedes hacerlo. Esperamos vuestras cartas o vuestros mensajes vía *e-mail*. La dirección es la siguiente:

Revista DIVMANÍA
Sección correo
C/Alfonso Gómez, número 42, nave 1-1-2.
Código postal: 28037, Madrid.
gover@prensatecnica.com

Contenido CD-Rom



Nuestro CD trae en este número una serie de programas muy interesantes. Desde una demo de Tokenkai, la demostración de las posibilidades de DIV a nivel profesional, hasta una buena pléyade de programas *shareware* para todos los gustos, pasando, como no, por los juegos que han resultado ganadores de nuestro concurso DIV.

DEMOS

Tie Break Tennis 2

Este juego, llamado *Tie Break Tennis 2*, ofrece a los amantes del deporte de la raqueta la posibilidad de exhibir sus habilidades en ocho de los más importantes torneos de la Asociación de Tenistas Profesionales (ATP), que incluyen los cuatro que forman el "Grand Slam"(el Open de Estados Unidos, Wimbledon, Roland Garros y el Open de Australia), así como los de Sao Paulo, Johannesburgo, Tokio y Moscú. Cada uno de estos torneos tiene unos determinados requisitos de puntos ATP que el jugador ha de ir alcanzando si quiere escalar puestos en la clasificación, y para acceder al siguiente torneo se deben lograr una cierta cantidad de puntos. Los torneos tienen tres rondas clasificatorias (cuartos de final, semifinales y la final), y por cada encuentro ganado, el jugador recibirá puntos ATP en función de la ronda superada.

America´s toughest 18

Este es un videojuego diseñado tanto para el disfrute de los aficionados al golf como para los que este deporte era, hasta el momento, algo completamente desconocido o simplemente ajeno. Se trata de disfrutar del desafío que representa el recorrido por los 18 hoyos considerados como los más difíciles de los campos de los Estados Unidos, país en el que el golf tiene una popularidad y un "status" como en casi ninguna otra parte del mundo. Cada hoyo es realmente el que corresponde a cada circuíto, lo que confiere al juego un aliciente especial.



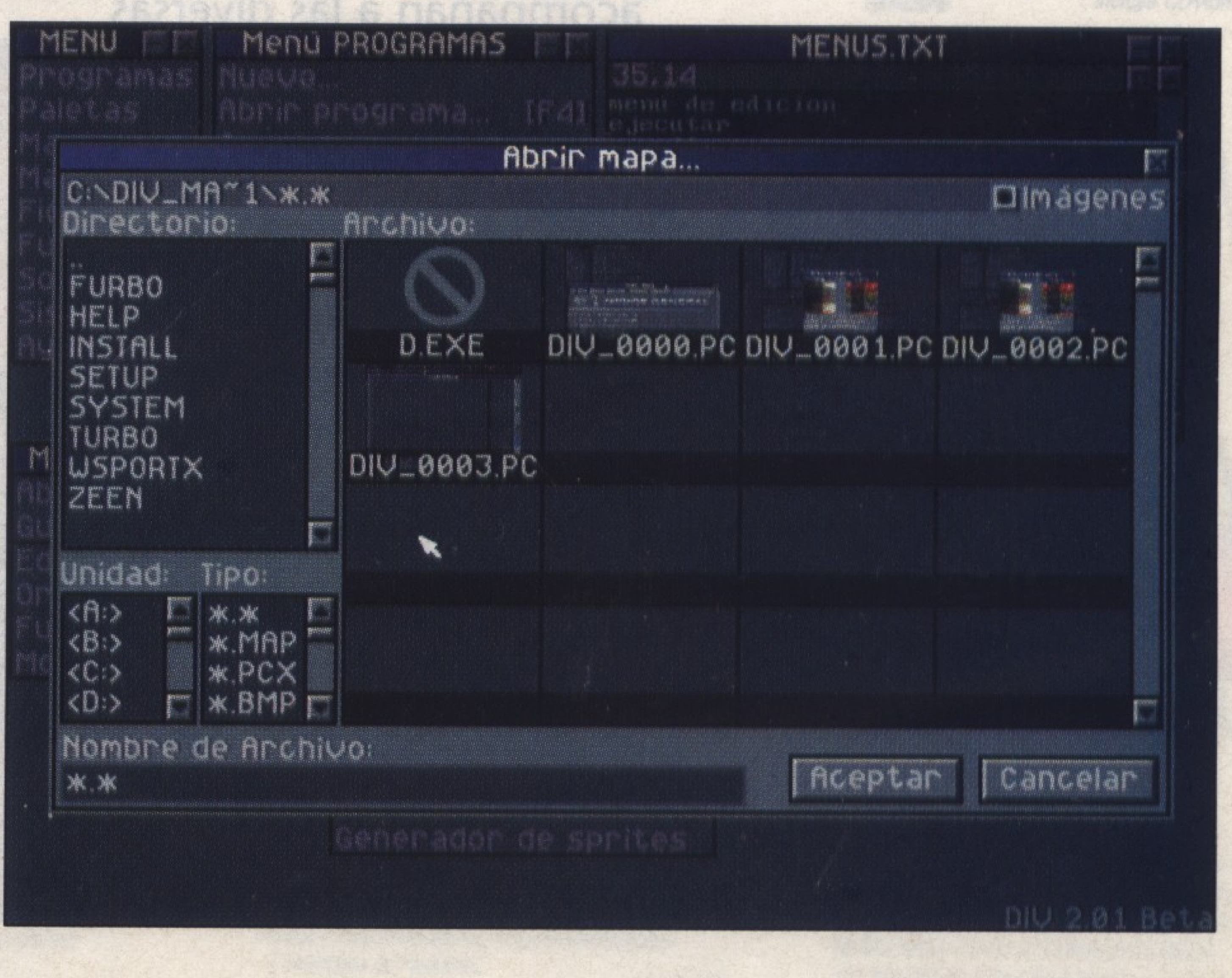
DIV 2

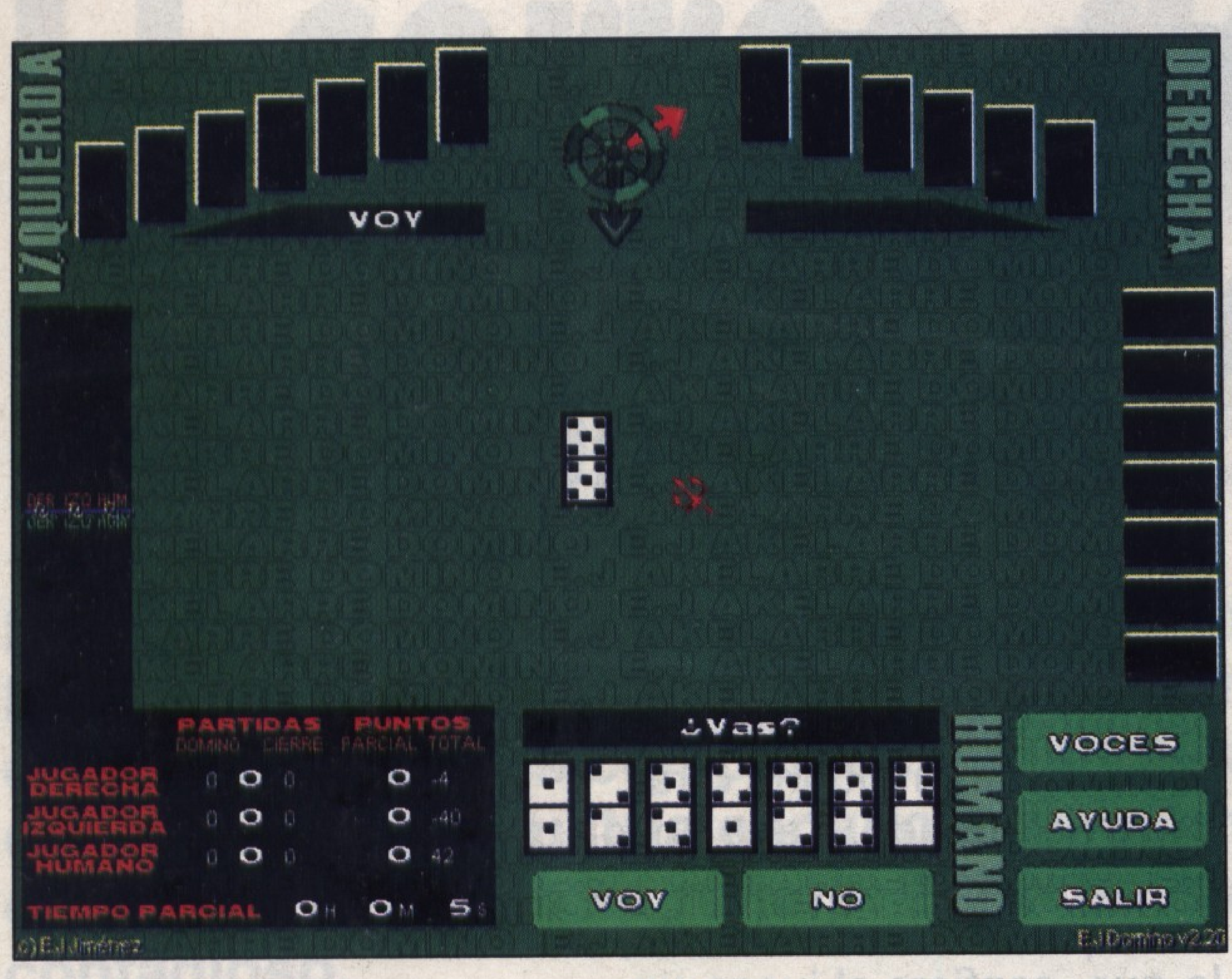
Por si hay algún despistado que todavía no ha probado la segunda versión de esta maravillosa herramienta para la programación de videojuegos de una manera fácil y sencilla, en este número volvemos a incluirla. Seguro que más de uno nos lo agradecerá.

JUEGOS COMPLETOS

Mr. Bones (o vete al infierno)

Os ofrecemos el título que se ha alzado con la victoria en el concurso de creación de juegos con DIV. Se trata de un divertido juego del género de las plataformas en el que manejaremos un tipo, algo flaco, que no para de lanzar huesos para acabar con los enemigos que le salen al paso. Un juego con una gran calidad gráfica que merece sin duda el galardón concedido.





Acelerator

El segundo juego que merece estar en estas páginas es este título de carreras de coches. Son especialmente divertidas las partidas a dos jugadores. En ellas la pantalla se divide en dos partes, cada jugador ve donde se encuentra en cada momento y los avances o dificultades de su contrincante en la carrera.



E. J. Dominó

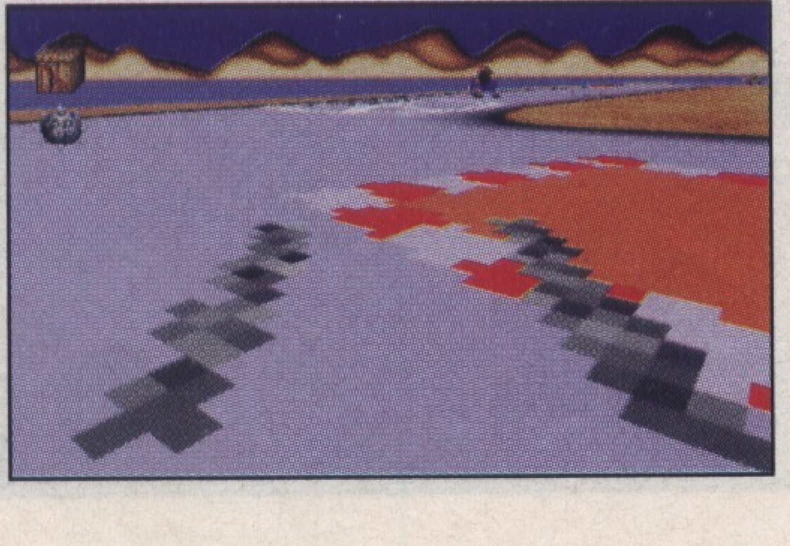
Si no tienes a nadie con quién jugar al dominó y eres un apasionado de este juego, aquí tienes un simulador virtual de este famoso juego de mesa. Jugarás contra dos jugadores imaginarios controlados por la CPU de tu ordenador. Este es el argumento del juego que se ha hecho con el tercer puesto de nuestro concurso de juegos programados por los lectores.

CURSOS DE JUEGOS

Nuestro CD también incluye los pequeños programas que acompañan a las diversas secciones de nuestra revista: aventura, rol, DIV, etc.

REVISTA DIVnet

Os ofrecemos la posibilidad de darle una ojeada a esta revista electrónica que podéis encontrar en Internet. La hemos encajado en el CD porque habrá muchos que no dispongan de conexión a la Red.



SHAREWARE

ACDSee 2.41

Uno de los más útiles visores en su última versión shareware.

CD Box Labeler 1.1

Te permite crear etiquetas y libretos para CDs.

Copernic 99 301

Este agente de software gratuito encontrará con exactitud lo que estés buscando en Internet.

Cyber-Info Webmail

Hotmail Notify es un sencillo programa que te avisa cuando tu cuenta de HotMail tiene nuevos mensajes.

Download Accelerator 3.5

Download Accelerator acelera la recepción de ficheros.

Eudora Pro 4.2

El cliente de correo electrónico más popular para Windows 95/NT.

FTP SERV-U 2.5ª Build 4

Un excelente lector y buscador dentro de newsgroups.

Hyper Maker

Aplicación que te permitirá transformar páginas Web en publicaciones comprimidas y encriptadas, para su distribución.

mIRC 2.01

Es uno de los mejores clientes de IRC, además de ser todo un clásico.

Nendo 1.1

Nendo es una potente herramienta de modelaje en 3D que, sin embargo, destaca por su facilidad de uso.

NetInfo 3.4

Programa multitarea totalmente configurable.

News Rover 4.42

Útil lector de news, gracias al cual podremos organizarlas de forma cómoda.

Organizador de Música

Esta excelente aplicación pondrá orden donde antes sólo había caos.

PACT JumpReg 99.6

JumpReg te puede llevar hasta las palabras clave de más uso del Registro.

PPPshar Pro 1.51

Con PPPshar, podrás conectar todos los ordenadores de tu red a Internet usando una sóla conexión telefónica.

SmartFTP 1.0 Build

SmartFTP nos proporciona un interfaz de múltiples ventanas con soporte para arrastrar y soltar.

SynEdit 0.40 beta

SynEdit es un excelente editor de textos orientado a programadores.

UltraEdit Professional

Es más que un reemplazo del Bloc de Notas, es un editor de textos completísimo.

URLSenty 1.14

Escanea de forma instantánea tus páginas Web favoritas.

Viruscan 4.0.3

Uno de los más importantes antivirus del mercado nos llega en su última versión shareware.

WebAurora 98 1.01

El editor Web para Windows 95 y 98.

WinAmp 2.22

Con este programa podremos escuchar música con la más alta fidelidad.

Winzip70SR-1

El más importante compresor del mercado.

WorkTime 1.1

Con Worktime puedes controlar cuánto tiempo pasas delante del ordenador trabajando, ya que te avisa cuando tu jornada ha terminado. Cuenta con dos tipos de alarma (normal y de cuenta atrás).

CONTENIDO DEL CD ROM



DEMO TIE BREAK TENNIS 2
Os ofrecemos la demo de un juego de tenis que ofrece a los amantes del deporte de la raqueta la posibilidad de competir en hasta ocho de los más importantes torneos de la Asociación de Tenistas Profesionales (ATP).

AMERICA´S TOUGHEST 18
Este es un videojuego diseñado tanto para el disfrute de los aficionados al golf como para los que este deporte era, hasta el momento, algo completamente desconocido o simplemente ajeno.

DIV 2
Volvemos a ofrecer la Demo de DIV2 por si hay algún despistado que todavía no la ha probado.

REVISTA DIVNET
Incluimos el número 0 de la primera revista electrónica dedicada exclusivamente a DIV. Echadle un vistazo porque realmente merece la pena.

JUEGOS GANADORES
Como no, os ofrecemos los juegos ganadores de nuestro concurso de programación con DIV: *Mr. Bones (o vete al infierno)*, un divertido programa de plataformas, *Acelerator*, un juego de carreras de karts y E. J. Dominó, un programa para los amantes del conocido juego de mesa.

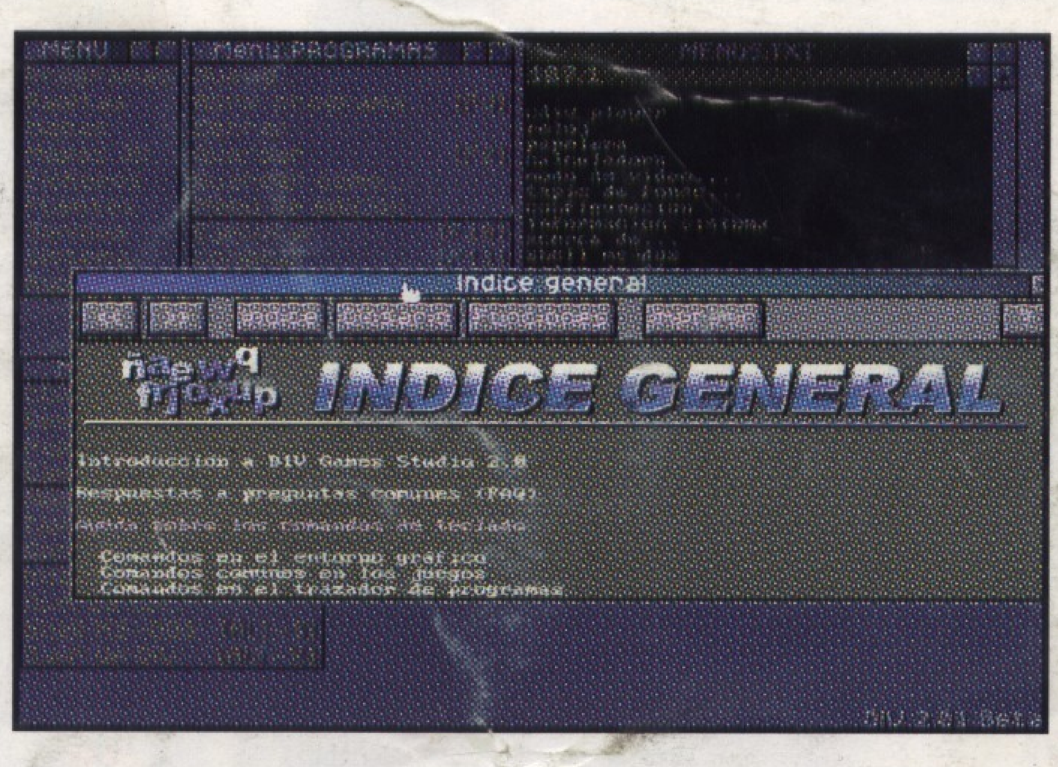
CURSOS DE JUEGOS
Nuestro CD también incluye los pequeños programas que acompañan a las diversas secciones de nuestra revista: aventura, rol, DIV, etc.

SHAREWARE
Como siempre, incluimos un buen número de programas shareware que, a buen seguro, os serán de utilidad: CD Box Labeler 1.1, Copernic 99 301, Cyber-Info, Webmail, Download Accelerator 3.5, Eudora Pro 4.2, FTP SERV-U 2.5ª Build 4, Hyper Maker, mIRC 2.01, Nendo 1.1, NetInfo 3.4, News Rover 4.42, Organizador de Música PACT JumpReg 99.6, PPPshar Pro 1.51, SmartFTP 1.0 Build, SynEdit 0.40 beta, UltraEdit Professional, URLSenty 1.14, Viruscan 4.0.3, WebAurora 98 1.01, WinAmp 2.22, Winzip70SR-1, WorkTime 1.1.

REPORTAJE. Un interesante reportaje imprescindible para cualquier programador.



DEMO DIV2. La demo de la segunda versión del entorno de juegos.

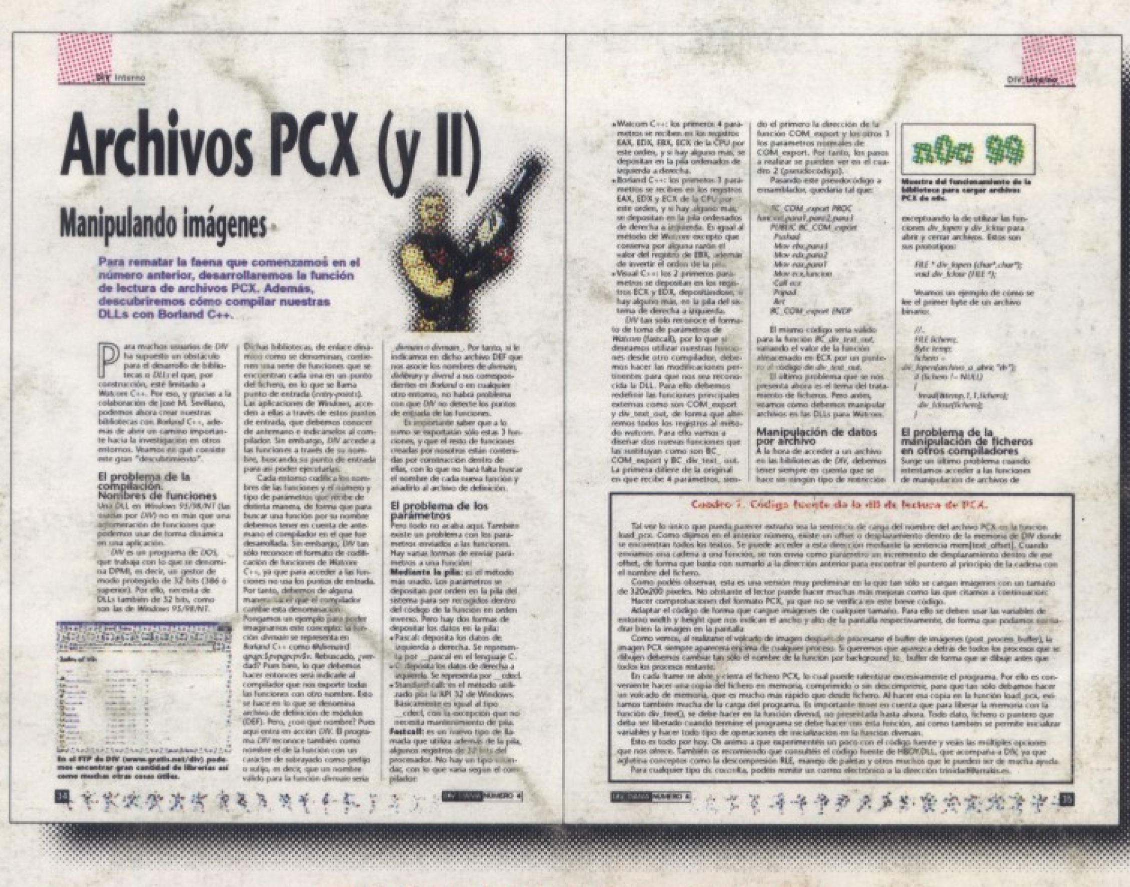


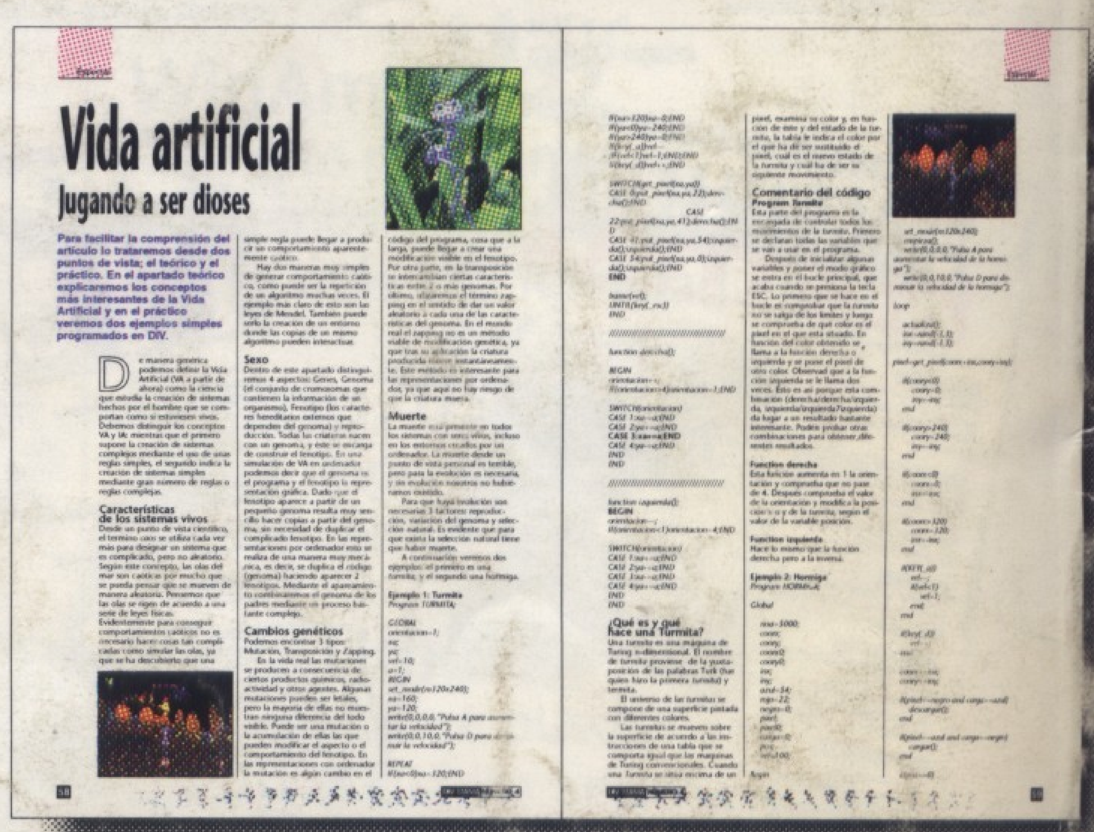
PROGRAMA DEL LECTOR. El resultado de nuestro concurso de programación de juegos.



CON EL MEJOR CONTENIDO







ACTUAL EXHAUSTIVO DIDÁCTICO

Y MUCHO MÁS...